كلية العلوم بتونس
FACULTE DES SCIENCES DE TUNIS

Vérification formelle de la relation de raffinement des diagrammes de séquence d'UML2.X avec la méthode B événementiel

THÈSE

Présentée et soutenue publiquement le 30 juin 2018

pour l'obtention du

Doctorat en Informatique

(mention informatique)

par

Fatma Dhaou

Composition du jury

Président :	M. Sadok Ben Yahia Professeur à la Faculté des Sciences de Tunis
Rapporteurs :	M. Samir Ben Ahmed Professeur à la Faculté des Sciences de Tunis Rapporteur : M. Jean-Paul Bodeveix Professeur à l'Université de Toulouse
Examinateurs :	Mme. Leila Ben Ayed Jemni Professeur à l' École Nationale des Sciences de l'Informatique Manouba
Encadrants :	M. Khaled Bsaies Professeur à la Faculté des Sciences de Tunis M. J. Christian Attiogbé Professeur à l'Université de Nantes Mme Inès Mouakher Abdelmoula Maître assistante à la Faculté des Sciences Économiques et de Gestion de Tunis

Thèse préparée dans le cadre d'une codirection
Fst-Tunis, (Laboratoire LIPAH) et (Laboratoire LS2N) Nantes France

Mis en page avec la classe thesul.

Remerciements

Je tiens à exprimer ma très profonde gratitude à M. Khaled Bsaies, Professeur à la Faculté des Sciences de Tunis, pour la confiance qu'il m'a accordée en acceptant de diriger ma thèse.
Je le remercie pour son souci constant de l'avancement de ma thèse et ses précieux conseils qui ont éclairé mon parcours.

J'adresse également mes plus vifs remerciements à M. J. Christian Attiogbé, Professeur à l'Université de Nantes, pour la confiance qu'il m'a témoignée dans la conduite de mes travaux. Je le remercie pour la disponibilité dont il a toujours fait preuve à mon égard, pour le soutien qu'il m'a apporté tout au long de ces années et son accueil chaleureux dans le laboratoire LS2N. Je suis fière de l'avoir eu comme *mentor* et d'avoir appris à ses côtés la rigueur scientifique et la pédagogie pour présenter et rédiger les travaux pendant ces années de thèse. Je ne le remercierai jamais assez et je lui serai toujours reconnaissant.

J'exprime ma sincère reconnaissance à Mme Inès Mouakher, Maître Assistante à la Faculté des Sciences Économiques et de Gestion de Tunis, pour les fructueuses discussions que nous avons eues. Elles ont sans aucun doute influencé mes travaux de recherche.
Je la remercie pour son suivi continu de mon travail. Ses conseils, ses encouragements ainsi que la confiance qu'elle m'a toujours témoigné m'ont été d'un grand apport tout au long de mes travaux. Je n'oublierai jamais que c'était grâce à elle que je fais de la recherche aujourd'hui.

Je remercie M. Samir Ben Ahmed, Professeur à la Faculté des Sciences de Tunis et M. Jean-Paul Bodeveix, Professeur à l'Université de Toulouse d'avoir accepté de rapporter mon travail.

Je remercie toutes les personnes qui ont contribué de près ou de loin à l'aboutissement de cette thèse.

Je dédie cette thèse

À la mémoire de mon cher papa avec lequel je n'aurais pas le plaisir de partager cet événement, mais qui est et qui demeurera dans mon cœur et à jamais. J'espère que je saurai à la hauteur des valeurs que tu as semé en nous. Papa la vie sans toi est fade.

À ma douce maman : ce travail est le fruit de tes innombrables sacrifices, j'espère que ta bénédiction m'accompagne toujours,

À mon frère Khaled et ma sœur Imen,

À mon mari : merci d'être toujours à mes côtés, par ta présence, pour ton dévouement et pour ta patience,

À mes enfants Rayan et Adam : je vous souhaite un avenir radieux,

À ma grande famille.

Sommaire

Table des figures

Chapitre 1

Introduction générale

1.1 Contexte

L'élaboration des spécifications est une phase très importante dans le processus de développement des systèmes étudiés. Les spécifications résultent de la synergie d'intervenants de diverses spécialités (des utilisateurs, des concepteurs, des développeurs, etc). Ces intervenants déploient des efforts pour élaborer une multitude de documents tels que le cahier des charges, les spécifications, les documents de conception, les documents de tests, les codes, etc.
Avoir des spécifications non ambiguës et complètes est primordiale. Dans la pratique industrielle, il y a souvent recours à des langages de spécification intuitifs et des techniques de développement, de vérification et de validation adéquates. Nos travaux de thèse s'inscrivent dans cette optique, plus précisément nous nous intéressons à la modélisation et à la spécification des comportements des systèmes distribués en suivant une technique de développement incrémentale, avec l'utilisation conjointe des diagrammes de séquence (DS) d'UML2.X et de la méthode formelle B événementiel.

1.2 Systèmes distribués ou répartis

La nature distribuée intrinsèque aux tâches de calcul modernes, ainsi que les inconvénients des systèmes centralisés, procurent aux systèmes distribués une popularité accrue. Les systèmes distribués sont définis comme un ensemble de ressources de différents types qui sont reliés par un réseau de communication.

Les systèmes distribués avant tout conçus pour répondre à la distribution géographique de certains systèmes existants, aux exigences de qualité de service des applications telles que la sécurité, la fiabilité, la disponibilité, l'extensibilité ou le temps de réponse, le partage des ressources, l'ouverture, la concurrence, la transparence et la tolérance aux fautes.

Cependant, il y a plusieurs problèmes qui sont liés à l'utilisation de ces systèmes tels que les problèmes qui sont inhérents à la communication entre les composants qui sont indépendants et autonomes. En effet la communication est majoritairement asynchrone. Le partage des ressources et des données pose un problème de synchronisation vue l'absence d'un état global instantané.

En outre la modélisation et la spécification de tels systèmes sont extrêmement difficiles. Les méthodes de spécification, de conception et d'analyse sont dotées de plusieurs aspects (l'encapsulation, la restructuration, la réutilisation et l'abstraction de données), qui font d'elles une piste privilégiée adoptée par les concepteurs pour la spécification et la modélisation des systèmes distribués. Beaucoup de méthodes ont émergé entre 1990 et 1995, mais aucune d'entre elles ne s'est vraiment pas imposée. Parmi ces méthodes, nous citons celles qui ont le plus influencé la modélisation objet au début des années 90 : la méthode *Booch* (qui est une méthode d'analyse et de conception orientée objet qui porte le nom de son inventeur) [Gra90], la technique de modélisation objet (*object-modeling technique (OMT)*) [Rum96] et le langage de modélisation

objet (*object oriented software engineering (OOSE)*) [Iva92].
UML (*Unified Modeling Language*) est né de la fusion de ces trois méthodes et a remporté une vaste popularité et il est devenu une norme de facto de l'industrie.

1.3 Problématique

Dans cette thèse, nous nous intéressons à la modélisation et la spécification des comportements des systèmes distribués, qui est reconnue comme un problème complexe. Généralement les travaux de l'état de l'art reposent, soit sur des formalismes formels, soit sur des formalismes basés sur les scénarios ou encore sur une approche hybride combinant les deux formalismes.

Les méthodes formelles contribuent essentiellement à l'amélioration des critères de fiabilité et d'intégrité. Elles visent à limiter les erreurs de spécification et de conception, en apportant à la fois des techniques visant à construire de manière correcte à *priori*, et des techniques de vérification pour analyser des failles à *posteriori*. En outre elles permettent l'expression et la vérification des propriétés importantes des systèmes. Cependant, en dépit de leur rigueur étant donné qu'elles sont basées sur des fondements mathématiques, elles restent difficiles à adopter surtout dans des étapes précoces de modélisation nécessitant l'intervention des experts. Les formalisations mathématiques posent des difficultés de validation par les clients pendant les phases initiales qui sont importantes et qui décident du succès ou de l'échec d'un projet. Ces limites réfutent le recours aux formalismes formels en amont.

Les approches basées sur les scénarios tels que UML, font partie des formalismes semi-formels ; elles ne rebutent pas les non-spécialistes de méthodes formelles. Elles sont attractives grâce à la convivialité de leurs représentations graphiques offrant une intuition de modélisation. Cependant, leur caractère semi-formel se heurte à des problèmes de vérification, qui ne peuvent être appréhendés autrement que par le couplage avec une méthode fiable basée sur des fondements mathématiques.

Par conséquent, pour tirer profits des avantages des méthodes semi-formelles et des méthodes formelles, nous avons opté pour une approche hybride couplant les deux formalismes. En adoptant le formalisme semi-formel les diagrammes de séquence d'UML2.X dans une phase préliminaire puis en intégrant la méthode formelle B événementiel dans une phase en aval.

La popularité croissante des diagrammes de séquence d'UML2.X revient à leurs représentations graphiques faciles, le grand pouvoir d'expression grâce à la richesse de ce langage avec l'incorporation de plusieurs aspects tels que les fragments combinés, permettent la modélisation des comportements complexes (les choix multiples, la concurrence, les itérations, etc). En outre ce langage est unifié et il peut être exploité durant toutes les phases de développement de logiciels, de l'expression des besoins jusqu'à la programmation en passant par la conception. L'*object management group (OMG)* [Obj15] a défini pour les diagrammes de séquence d'UML2.X une syntaxe précise, en revanche la sémantique standard souffre des ambiguïtés émanant des définitions fixées pour les calculs des traces valides possibles. En effet, elles ne sont pas adéquates pour tous les types des systèmes.

Notre choix de la méthode formelle B événementiel se justifie, non seulement par la rigueur de cette méthode qui est comme toute méthode formelle basée sur des fondements mathématiques, mais en plus par son environnement puissant offrant des outils de vérification de preuves et des outils d'animations automatiques ou interactifs. Les outils permettent de détecter ou d'éviter un certain nombre d'erreurs de spécification et de conception et ils supportent une activité de raffinement. Notons que peu d'environnements offrent des outils couvrant ces aspects.

Une construction incrémentale des spécifications est naturelle et pertinente. Elle permet de raisonner sur un problème de modélisation complexe et de valider les spécifications pas à pas. Elle vise l'élaboration des spécifications par étapes successives permettant, à partir d'une

spécification abstraite et incomplète, d'aboutir à une spécification complète ayant un niveau de détail satisfaisant et implémentable.

Par conséquent, nous avons appliqué une construction incrémentale des modèles comportementaux des systèmes distribués en utilisant les diagrammes de séquence d'UML2.X. La validation de chaque diagramme obtenu par raffinement nécessite la formalisation d'une relation de raffinement. Les relations de raffinement ne sont pas bien définies sur les diagrammes de séquence d'UML2.X. Cependant elles sont bien définies sur les sémantiques opérationnelles. Par conséquent, les parties que nous avons cerné pour les traiter dans nos travaux se résument comme suit :

- définir une sémantique opérationnelle appropriée aux diagrammes de séquence d'UML2.X qui modélisent les comportements des systèmes distribués,
- étudier le raffinement des diagrammes de séquence d'UML2.X en définissant clairement des règles qui régissent un raffinement correct,
- formaliser une relation de raffinement qui possède des propriétés favorables à un développement incrémental,
- traduire les diagrammes de séquence d'UML2.X vers des spécifications en B événementiel : proposer un modèle générique de traduction,
- valider et vérifier les spécifications B avec les outils dédiés : prouveur de théorèmes (Rodin) et explorateur des modèles (*model-checker ProB*).

1.4 Contributions

Dans le but de définir une sémantique opérationnelle pour les diagrammes de séquence d'UML2.X modélisant les comportements des systèmes distribués qui va être la base de la vérification de la relation de raffinement entre des diagrammes de séquence d'UML2.X, nous avons rapporté des faiblesses de la sémantique standard. Un problème fondamental est la diversité des interprétations des diagrammes de séquence par les sémantiques existantes même pour les diagrammes de séquence basiques. Dans le cadre de nos travaux nous avons constaté l'inadéquation des définitions de la sémantique standard pour le calcul des relations de précédence (ordre partiel) entre les événements pour un diagramme de séquence modélisant des interactions entre des composants distribués.

Pour pallier ce problème, nous avons considéré une sémantique existante, qui est la *sémantique causale*. Néanmoins, cette sémantique a été proposée pour les diagrammes de séquence (DS) d'UML1.X basiques. Notre première contribution est d'étendre les relations de cette sémantique pour supporter les diagrammes de séquence d'UML2.X dotés de structures de haut niveau, qui sont les fragments combinés (FC). Nous avons considéré dans un premier temps les fragments combinés les plus utilisés (ALT, OPT, LOOP) permettant de modéliser respectivement des comportements alternatifs, optionnels et répétitifs. Les diagrammes de séquence comportant ces fragments combinés nécessitent un traitement méticuleux pour la détermination des relations de précédence entre les événements.
Ensuite, nous avons proposé une deuxième extension pour généraliser les lois de la sémantique causale pour qu'elles permettent de traiter les FC combinés imbriqués. Nous avons également considéré d'autres FC importants (PAR et STRICT) permettant de modéliser respectivement des comportements concurrents et stricts et nous avons traité certains problèmes liés aux FC.

En se basant sur la sémantique causale qui est fondée sur la théorie de l'ordre partiel, nous avons défini une sémantique opérationnelle.
La troisième contribution de notre travail consiste à la définition des raffinements des diagrammes de séquence d'UML2.X et des lois qui régissent un raffinement correct entre des diagrammes de séquence considérés modélisant des comportements d'un système distribué. La sémantique opérationnelle que nous avons préliminairement définie, a servi comme base pour la formalisation de la relation de raffinement.

La quatrième contribution de notre travail consiste en la traduction de la sémantique opérationnelle vers la méthode formelle B événementiel pour la vérification, la validation de notre sémantique opérationnelle et de la relation de raffinement que nous avons établie. Des propriétés de sûreté, de vivacité et temporelles ont été également exprimées et vérifiées.

1.5 Organisation de la thèse

Cette thèse est constituée d'une introduction générale, de six chapitres et d'une conclusion générale.

Dans l'introduction générale, nous présentons le contexte, les motivations et les problématiques de recherche abordées et les contributions issues de nos travaux.

- le **deuxième chapitre** intitulé "*les diagrammes de séquence d'UML2.X : état de l'art*" introduit les notions de base de ces diagrammes leurs syntaxes et sémantiques. Nous montrons la richesse de ce langage qui procure la possibilité de la modélisation de différents types de comportements sophistiqués des systèmes modélisés. Nous avons également présenté un état de l'art des sémantiques existantes qui sont proposées pour ces diagrammes et nous avons souligné quelques problèmes qui sont liés à l'utilisation de ces diagrammes.
- le **troisième chapitre** intitulé "*la sémantique causale et ses extensions*" présente d'abord les principes de base de la *sémantique causale* qui est dédiée aux diagrammes de séquence d'UML1.X qui modélisent les comportements des systèmes distribués. Puis nous avons montré les insuffisances de cette sémantique pour les diagrammes de séquence d'UML2.X. Ensuite, nous avons proposé dans un premier temps une première extension de cette sémantique qui consiste d'abord à la proposition d'une formalisation des diagrammes de séquence d'UML2.X, puis à l'extension des relations de cette sémantique pour supporter quelques FC les plus populaires en émettant des hypothèses syntaxiques assez restrictives (imposer une disposition particulière des FC en séquentiel), en adoptant une interprétation non-standard des FC, restreindre la profondeur d'imbrication des FC, etc.).
  Dans un second temps, nous avons proposé une deuxième extension de cette sémantique qui vise à généraliser les relations de la sémantique causale afin de supporter plus de FC et en relâchant les hypothèses syntaxiques en adoptant leurs interprétations standard et en permettant l'imbrication des FC à n'importe quelle profondeur.
- le **quatrième chapitre** intitulé "*sémantique opérationnelle*" est dédié à définir et formaliser le comportement d'un diagramme de séquence d'UML2.X et les stratégies des occurrences de ses événements.
- dans le **cinquième chapitre** intitulé "*Raffinement des diagrammes*", nous précisons comment le processus de raffinement est appliqué aux diagrammes de séquence d'UML2.X, nous énonçons de façon informelle les règles qui régissent un raffinement correct puis nous formalisons la relation de raffinement.
- dans le **sixième chapitre** intitulé "*Implémentation de la sémantique opérationnelle en B événementiel*", nous présentons notre approche pour l'élaboration d'un modèle générique de traduction des diagrammes de séquence d'UML2.X vers des spécifications en B événementiel. Les DS considérés sont équipés de la sémantique opérationnelle définie dans le quatrième chapitre. Nous vérifions la cohérence de notre modèle B ainsi que quelques propriétés avec les outils de l'environnement de la méthode B événementiel (le prouveur de théorèmes et l'explorateur de modèles (*model checker*)).
- dans le **septième chapitre** intitulé "*Implémentation de la relation de raffinement des diagrammes de séquence d'UML2.X en B événementiel*, nous étendons le modèle générique de traduction des diagrammes de séquence d'UML2.X vers des spécifications en B le en proposant un modèle générique de vérification de la relation de raffinement des diagrammes de séquence d'UML2.X. Nous illustrons notre approche avec des expérimentations.

Dans la **Conclusion générale**, nous clôturons notre mémoire dressant le bilan de nos contri-

butions et en présentant nos perspectives de recherche.

Chapitre 2

Les diagrammes de séquence UML2.X : état de l'art des sémantiques

Sommaire

2.1 Introduction

Le langage de modélisation unifié UML est décrit comme un langage universel graphique de modélisation objets qui est adopté par le standard *object management group (OMG)* [Obj15]. Ce langage définit des modèles qui offrent une vue abstraite des systèmes, permettant de les représenter selon des niveaux de détails différents. Un modèle peut englober des éléments tels que des acteurs, des cas d'utilisations, des classes, des packages, un ou plusieurs diagrammes qui montrent une perspective spécifique d'un système. Pour mettre en œuvre ces modèles, UML propose 14 diagrammes qui peuvent être catégorisés selon les vues qu'ils permettent de décrire.

- vues du système : statique et dynamique,
- les $4+1$ vues d'architecture : c'est une vue modèle qui a été proposée par [Phi95] (Figure 2.1)
  - vue utilisateur : représente les besoins et les finalités exigées des clients du système,
  - vue structurelle : représente les éléments de structure pour la mise en place d'une solution pour les besoins définis,
  - vue comportementale : représente les aspects dynamiques du comportement du système,
  - vue implémentation : représente les aspects de la structure et du comportement du système,
  - vue environnementale : représente les aspects de la structure et du comportement du domaine dans lequel la solution est réalisée.

Les diagrammes de séquence (DS) sont une instance possible des diagrammes d'interactions ;

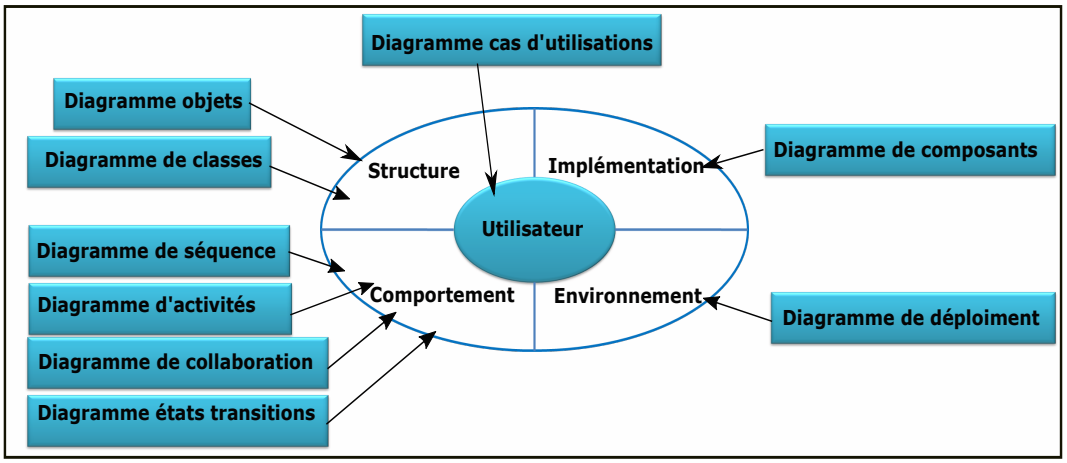

FIGURE 2.1 – 4 + 1 vues d'architecture

ils permettent de capturer les comportements dynamiques du système en représentant les interactions entre ses composants dans un enchaînement temporel.

2.2 Concepts de base des diagrammes de séquence UML2.X

Les diagrammes de séquence d'UML2.X sont dotés d'une syntaxe concrète (voir Figure 2.2) et d'une syntaxe abstraite (voir Figure 2.3). La syntaxe concrète est la représentation graphique des éléments qui le constituent, la syntaxe abstraite est donnée sous la forme d'un méta-modèle qui précise les relations existantes entre les éléments graphiques.

2.2.1 Syntaxe Concrète

Un DS est fondamentalement constitué par des lignes de vie et des messages. Les lignes de vie sont graphiquement représentées par des lignes verticales permettant de modéliser concrètement des composants du système ou des instances d'objets. Les lignes de vie interagissent en échangeant des messages. Les extrémités de chaque message sont appelées des spécifications d'occurrence et correspondent à l'envoi du message et à sa réception, on les note avec ! et ? comme préfixe du message par (exemple pour un message m : $!m$, $?m$). Un message modélise une communication synchrone ou asynchrone. Durant une communication synchrone, l'expédi-

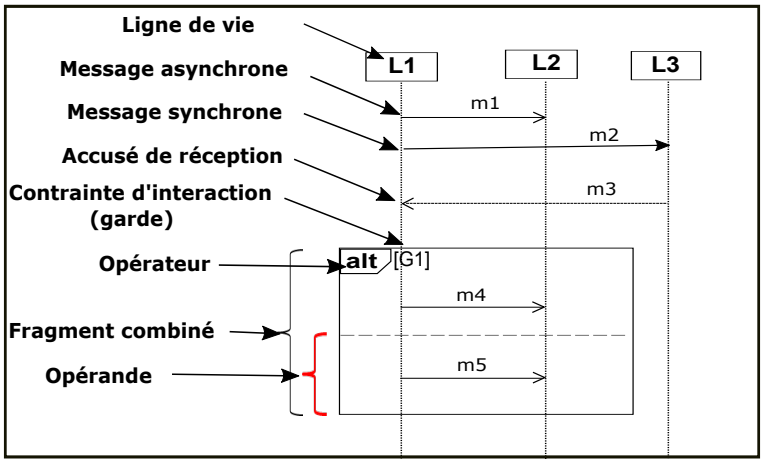

FIGURE 2.2 – Syntaxe concrète du diagramme de séquence

teur reste bloqué jusqu'à obtention d'un accusé de réception (qui est graphiquement représenté par une ligne en pointillé). Le flot de contrôle passe de l'émetteur au récepteur, toutefois le message peut être réflexif ; graphiquement un message synchrone est représenté par une flèche pleine, tandis qu'un message asynchrone est représenté par une flèche creuse. Les messages sont étiquetés par des noms, qui peuvent indiquer soit l'appel d'une méthode soit l'envoi d'un signal.

Avec la norme UML2.X, les DS ont emprunté de nouveaux aspects de leurs prédécesseurs, les diagrammes de séquence de messages (*message sequence charts* (MSC)), qui ont augmenté leur pouvoir d'expression permettant ainsi de modéliser plus de comportements complexes. Ces nouveaux aspects correspondant aux structures de haut niveau appelées *fragments combinés* (FC). Un FC est défini par un type d'opérateur, indiquant sa signification et par un ou plusieurs opérandes. Graphiquement, un FC est représenté par une boîte contenant un petit rectangle qui est situé dans le coin haut gauche dans lequel l'opérateur est indiqué. La boîte peut être subdivisée en plusieurs compartiments indiquant le nombre d'opérateurs qu'inclut le FC. Il existe douze opérateurs permettant de modéliser différents comportements du système étudié.
Dans la littérature, nous trouvons deux types de catégorisations possibles pour les FC. La première catégorisation est proposée dans les travaux de [Zol11] comme suit :
• les opérateurs qui permettent une représentation compacte du diagramme. Ce sont les fragments combinés ALT, OPT et LOOP, permettant de modéliser respectivement des comportements alternatifs, optionnels et itératifs,
• les opérateurs qui permettent de catégoriser les traces [1] en des traces valides ou invalides. Ce sont les fragments combinés NEG, ASSERT, IGNORE et CONSIDER, permettant de modéliser respectivement des comportements négatives, des assertions, des comportements insignifiants et des interactions à prendre en compte,
• les opérateurs qui influent sur l'ordonnancement des événements. Ce sont les fragments combinés PAR, CRITICAL et STRICT SEQUENCING, permettant de modéliser respectivement des comportements parallèles, une région critique et un séquencement strict entre les itérations.
Une deuxième catégorisation est proposée par [MSD] comme suit :
• les opérateurs des flux de contrôle : OPT, ALT, LOOP, BREAK, PAR, CRITICAL, SEQ, STRICT,
• les opérateurs d'interprétation de la séquence : CONSIDER, IGNORE, ASSERT, NEG.

Certains FC peuvent comporter des *contraintes d'interactions* appelées aussi *gardes*, qui peuvent être des expressions booléennes et/ou des prédicats de la logique du $1^{er} ordre$. Les gardes permettent de conditionner l'occurrence des événements qui se trouvent à l'intérieur du FC. Graphiquement, la garde est représentée entre crochets ([expression booléene]).
Nous pouvons trouver aussi dans un DS un invariant d'état. Graphiquement, il est représenté entre accolades ou dans un symbole qui est placé sur une ligne de vie. Il peut exprimer l'état de la ligne de vie et il doit être évalué avant l'occurrence de l'événement en cours sur la même ligne de vie. Des contraintes temporelles peuvent aussi figurer dans un DS.

2.2.2 Syntaxe abstraite

La syntaxe abstraite d'un diagramme de séquence est donnée par le méta-modèle qui est basé sur le diagramme de classe. Selon le standard *MOF* (*Meta Object Facility*) [OMG13], *un méta-modèle définit la structure que doit avoir tout modèle conforme à ce méta-modèle. Les méta-modèles fournissent les définitions des entités d'un modèle, ainsi que les propriétés de leurs connexions et leurs règles de cohérence.* Toutefois, un méta-modèle fournit peu de précision sur leurs sémantiques.

1. Une trace est une séquence possible d'occurrence d'événements

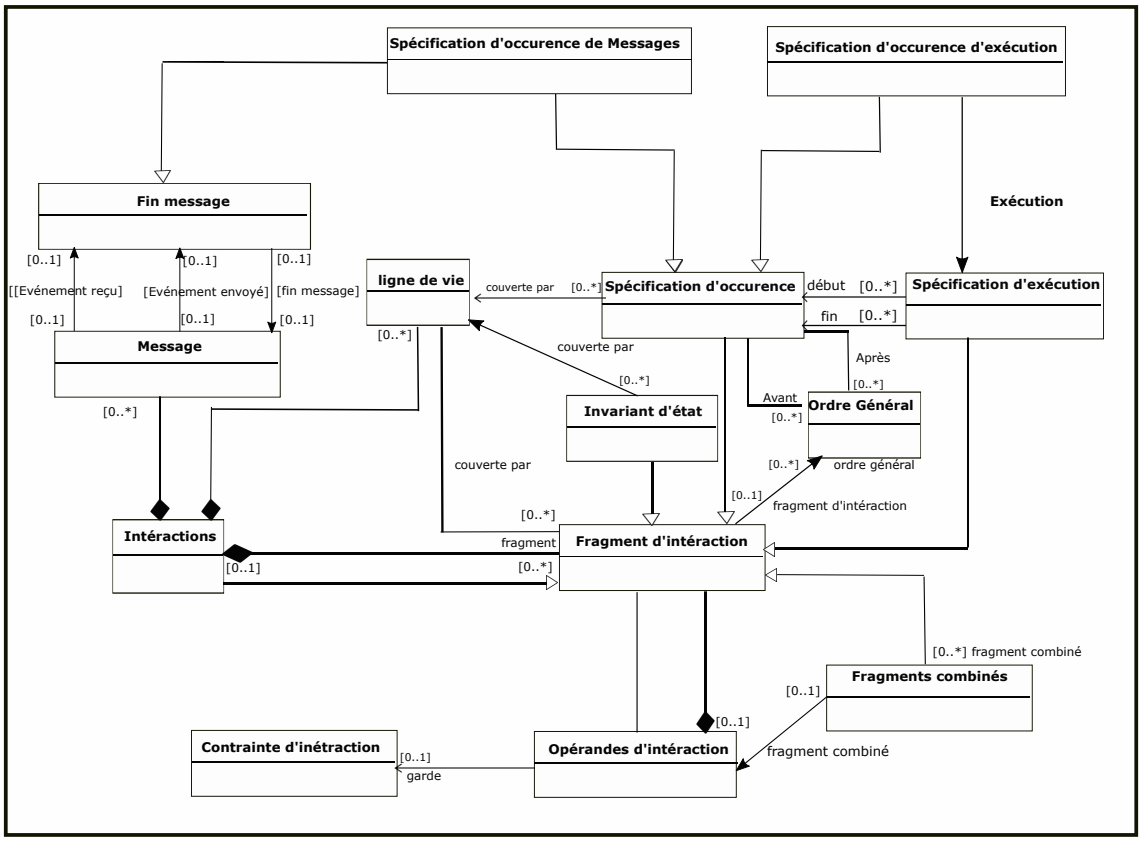

FIGURE 2.3 – Méta-modèle du diagramme de séquence

2.3 Sémantique des diagrammes de séquence UML2.X

Dans le domaine scientifique (mathématique, logique), la sémantique est le sens d'une notation. Généralement, le mot sémantique est employé par les développeurs pour parler du comportement [2] d'un système qu'ils développent. Pour UML, la sémantique est un moyen qui permet de comprendre comment ce langage doit être utilisé [Stu99]. Une définition de sémantique est composée de trois parties : la syntaxe, le domaine sémantique et la mise en correspondance entre la syntaxe et le domaine sémantique.
La définition d'une sémantique pour les diagrammes UML est un champ de recherche très actif, du fait de l'existence de quelques ambiguïtés persistantes de la sémantique standard. Ces ambiguïtés sont désignées par des *points de variations sémantiques* par certains auteurs [Har05], [Zol11], qui prétendent qu'elles sont intentionnelles en expliquant qu'elles offrent une grande flexibilité au langage UML pour l'adapter à la modélisation des systèmes de différents types.

2.3.1 Sémantique standard

Selon l'OMG, la sémantique d'un message est composée de <événement d'envoi, événement de réception>. La sémantique d'une interaction est donnée comme une paire d'ensemble de traces valides et invalides. Par conséquent l'élément principal de la sémantique est la trace, qui est composée par l'occurrence des événements. L'occurrence d'un événement dépend des conditions de déclenchement qui doivent être satisfaites au préalable.
Une condition nécessaire de déclenchement consiste à la satisfaction des contraintes de précédences, qui sont régies par un ensemble de définitions informelles fixées par le standard. La sémantique standard est une sémantique de traces, qui est basée à la fois sur la sémantique

2. Le comportement d'un système, modélisé par un diagramme de séquence, est défini par l'ensemble des traces que produit le système durant ses exécutions.

d'ordre partiel pour le calcul des traces pour les interactions basiques et sur la sémantique d'entrelacement (*interleaving-semantics*) pour le calcul des traces pour les fragments combinés : c'est à dire deux événements déclenchables ne peuvent pas s'exécuter en même temps ; de ce fait la sémantique standard ne supporte pas la vraie concurrence.

2.3.1.1 Sémantique d'un diagramme de séquence basique

Pour un diagramme de séquence basique, l'ensemble des définitions permettant de déterminer l'ordre partiel entre les événements échangés dans un DS est définit comme suit :
• chaque message doit être envoyé avant sa réception,
• les événements sont ordonnés le long de chaque ligne de vie.
Nous désignons par $!m$ et $?m$ respectivement l'envoi et la réception du message m. Le symbole $<$ désigne précède.

Illustration : considérons le diagramme de séquence de la Figure 2.4, nous spécifions les relations de précédence sur chaque ligne de vie du DS.
Sur la ligne de vie L1, nous avons $!m1 <!m4$; sur la ligne de vie L2, nous avons $!m2 < ?m3$.
Sur la ligne de vie L3, nous avons $?m1 < ?m2$ et sur la ligne de vie L4, nous avons $!m3 < ?m4$.

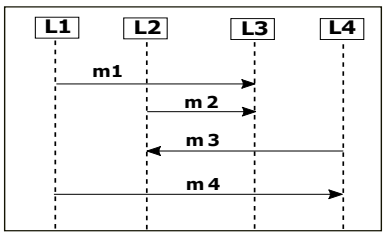

FIGURE 2.4 – Diagramme de séquence basique

2.3.1.2 Sémantique des fragments combinés :

1. le fragment combiné SEQ : cet opérateur est appelé souvent "faible séquencement" (WEAK SEQUENCING). Les lois de cet opérateur selon le standard OMG sont explicitées ci-dessous :
   • l'ordre de l'occurrence des événements à l'intérieur des opérandes est maintenu dans le résultat,
   • les événements sur les différentes lignes de vie des différents opérandes peuvent se déclencher dans n'importe quel ordre,
   • les événements sur la même ligne de vie des différents opérandes sont ordonnées tels qu'un événement du premier opérande se déclenche avant un événement du second opérande.
2. le fragment combiné ALT : ce FC permet de représenter des comportements alternatifs ; il peut comporter plus de deux opérandes. Un fragment combiné ALT peut comporter une contrainte d'interaction appelée aussi garde, qui peut être soit implicite soit explicite. Cette garde conditionne le déclenchement des événements qui sont couverts par l'opérande pour laquelle la garde est évaluée à *vrai*. Au plus un opérande est exécuté, de façon indéterministe, même en l'existence de plusieurs opérandes ayant des gardes qui sont simultanément évaluées à *vrai*. Le cas où aucune garde n'est évaluée à *vrai* est possible, dans ce cas aucun opérande n'est exécuté. L'ensemble des traces d'un fragment combiné ALT est défini comme l'union des traces de ses opérandes,
3. le fragment combiné OPT : ce FC est équivalent à un fragment combiné ALT ayant deux alternatives, telle que la deuxième alternative est vide,

4. le fragment combiné LOOP : il permet de modéliser un comportement itératif. Les événements couverts par le FC itèrent un nombre compris entre les valeurs min et max qui sont indiquées entre crochets. Le FC peut également comporter une garde, dans ce cas les événements itèrent tant que la garde est évaluée à $vrai$ et le nombre maximal d'itérations n'est pas atteint.
   Quand la garde est évaluée à $faux$, le LOOP s'arrête. Le fragment combiné LOOP est défini comme une application récursive de l'opérateur SEQ, c'est à dire l'opérateur faible séquencement est appliqué entre les itérations, de façon que l'occurrence de certains événements appartenant à différentes itérations peuvent se chevaucher,
5. le fragment combiné CRITICAL : le fragment combiné CRITICAL représente une section critique, l'occurrence des événements qui sont couverts par ce FC ne peuvent pas se chevaucher avec les autres événements du DS.
6. le fragment combiné PAR : ce FC représente une fusion parallèle entre les comportements de ses opérandes. Les événements des différents opérandes peuvent se chevaucher n'importe comment tels que l'ordre des événements dans chaque opérande est préservé.

   Illustration : les traces du DS comportant un fragment combiné PAR, qui est représenté par la Figure 2.6, sont :

$$\left\{ \begin{array}{l} (!m1, ?m1, !m2, ?m2), (!m1, !m2, ?m1, ?m2), (!m1, !m2, ?m2, ?m1), \\ (!m2, ?m2, !m1, ?m1), (!m2, !m1, ?m2, ?m1), (!m2, !m1, ?m1, ?m2) \end{array} \right\}$$

7. NEG : ce FC permet de modéliser des comportements négatifs et les traces résultantes sont considérées comme invalides.
8. ASSERT : le fragment d'opérande spécifie les seules séquences valides, ainsi toutes les autres séquences possibles sont invalides. Le fragment combiné ASSERT est généralement associé avec les fragment combiné CONSIDER ou IGNORE.
9. STRICT : le standard décrit le STRICT comme un opérateur avec plusieurs opérandes ; les opérandes sont exécutés dans l'ordre l'une après l'autre mais l'ordre de l'occurrence des événements dans chaque opérande est maintenu.

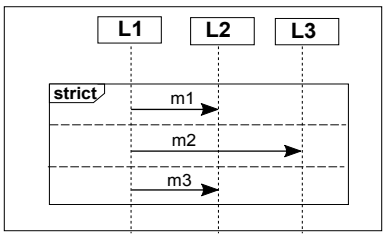

FIGURE 2.5 – Diagramme de séquence avec un FC STRICT

   Illustration : la trace du DS comportant un fragment combiné STRICT, qui est représenté par la Figure 2.5, est la suivante : $\{!m1, ?m1, !m2, ?m2, !m3, ?m3\}$
10. IGNORE : ce FC désigne des messages que peut produire le système durant son exécution et qui ne sont pas significatifs pour les lignes de vie couverts par le FC,
11. CONSIDER : contrairement au fragment combiné IGNORE, le fragment combiné CONSIDER désigne les messages qui doivent être considérés,
12. BREAK : ce FC est utilisé dans les fragments combinés qui représentent des scénarios d'exceptions. Les interactions de ce fragment seront exécutées à la place des interactions décrites en dessous. Le fragment combiné BREAK peut comporter une garde, dans ce cas il n'est exécuté que si la garde est évaluée à $vrai$, sinon le FC est ignoré.

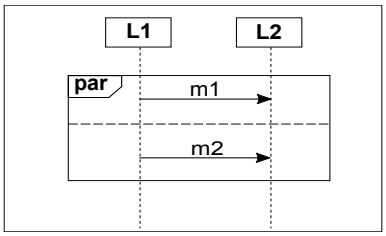

FIGURE 2.6 – Diagramme de séquence avec FC PAR

La trace résultante d'un diagramme de séquence comportant un FC est calculée en composant les traces obtenues des différents constituants (Ci), (comme illustré dans la Figure 2.7), et en les combinant par l'opérateur *faible séquencement* (SEQ) ; la sémantique standard est qualifiée de sémantique compositionnelle.

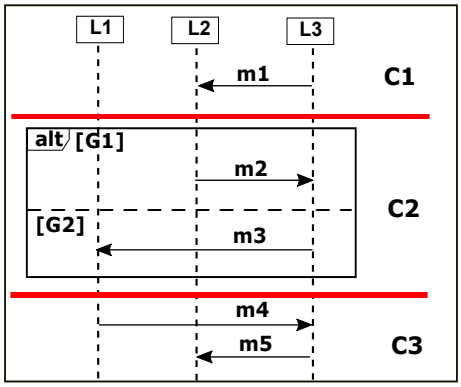

FIGURE 2.7 – Constituants d'un DS

2.3.1.3 Les modélisations possibles d'un diagramme de séquence

Un diagramme de séquence peut être utilisé pour modéliser des interactions qui varient selon les besoins du concepteur, en effet nous avons plusieurs interprétations d'un DS qui peuvent se résumer comme suit :
• un DS peut modéliser une ou quelques fonctionnalités du système, dans ce cas le DS modélise une vue partielle du système, seuls les composants impliqués dans l'interaction sont représentés,
• un DS peut modéliser tout le comportement du système, ainsi le DS comporte tous les composants du système,
• un DS peut servir à illustrer des exécutions possibles d'un système : la Figure 2.8 représente le comportement de tout le système, la Figure 2.9 représente quelques exécutions possibles du système,
• un DS peut servir à spécifier des propriétés de sûreté ou de vivacité.

2.3.2 État de l'art des sémantiques existantes

Les travaux de l'état de l'art ont adopté diverses approches (Figure 2.10) afin de définir une sémantique pour les modèles UML. Une catégorisation de ces approches est donnée comme suit [Kev99] :
• *la sémantique dénotationnelle* : elle consiste à exprimer les éléments du langage par des objets mathématiques (des fonctions),
• *la sémantique opérationnelle* : elle définit exactement les étapes d'exécution du langage avec des systèmes de transitions étiquetées (*labelled transition system* (LTS)). Ces sémantiques sont adaptées à la vérification formelle des propriétés sémantiques,

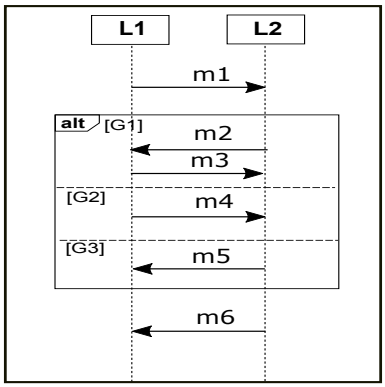

FIGURE 2.8 – Diagramme de séquence avec un fragment combiné ALT

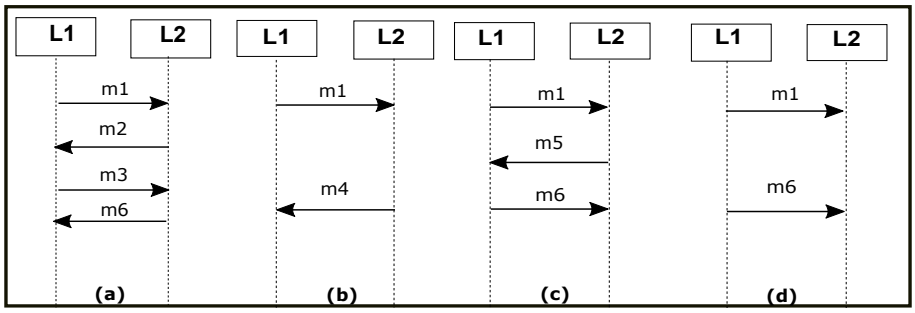

FIGURE 2.9 – Exécutions possibles du DS de la Figure 2.8

• *la sémantique transformationnelle* : elle consiste à fournir une sémantique pour les modèles UML en les traduisant vers des formalismes qui sont dotés d'une sémantique précise et bénéficier des résultats prouvés sur eux. Parmi les formalismes cibles les plus populaires nous citons les réseaux de pétri, les machines à états, la méthode B, etc...,
• *la sémantique axiomatique* : elle vise à définir une interprétation (sous forme d'axiomatisation) des modèles UML vers des formalismes mathématiques tels que la théorie des ensembles et la logique du premier ordre. Ce type de sémantique fournit un style adapté à la preuve des programmes par le moyen des assertions,
• *la sémantique méta-modèle* : elle utilise un sous ensemble d'UML comme un domaine sémantique d'UML [Gr07],
• *la sémantique co-algébrique* : elle consiste à utiliser des spécifications algébriques pour définir la sémantique. Notons qu'une approche sémantique proposée peut combiner deux approches parmi

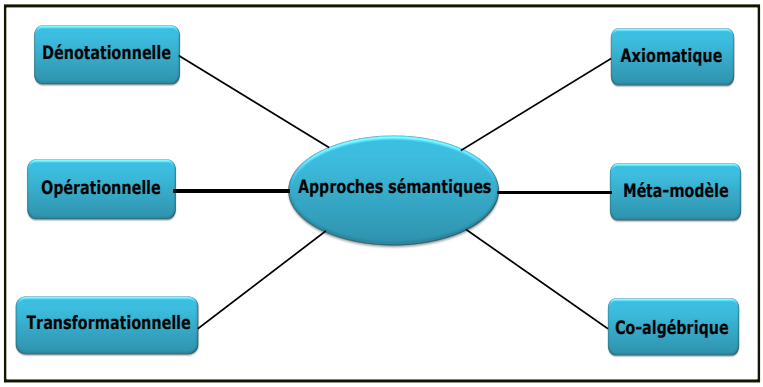

FIGURE 2.10 – Approches sémantiques

celles qui sont citées ci-dessus.

En définissant une sémantique, deux aspects importants doivent être précisés :
i) si elle supporte la concurrence ou non ; dans le premier cas la sémantique est appelée "*true concurrency*", autorisant le déclenchement simultané de deux événements distincts ; tandis que dans le deuxième cas, la sémantique est appelée sémantique d'entrelacement, permettant le déclenchement d'un seul événement à un instant donné ;
ii) si le temps est explicitement représenté, dans ce cas la sémantique est appelée "*branching time semantics*", sinon la sémantique est appelée sémantique linéaire.
Nous donnons un aperçu sur les travaux qui se sont intéressés particulièrement à définir une sémantique pour les diagrammes de séquence en précisant pour chacune l'objectif visé.

2.3.3 Approches sémantiques proposées pour les diagrammes de séquence

1. Les sémantiques dénotationnelles :
   dans les travaux de [Øy05], les auteurs ont défini une sémantique de traces qui est basée sur la sémantique dénotationnelle, dans laquelle ils ont considéré uniquement les traces valides (positives). La sémantique proposée couvre différents aspects des DS tels que les FC et les contraintes temporelles.

   Dans [Yo06], dans un but d'utiliser des outils automatiques pour l'analyse, la simulation et la vérification des modèles UML, les auteurs ont proposé une sémantique formelle pour les DS en utilisant la structure "*branching time*" et l'ordre partiel dans un style dénotationnel.
   La sémantique a été exploitée pour la vérification de la cohérence entre les diagrammes d'interaction d'UML par rapport aux diagrammes d'état. La sémantique proposée a été étendue pour supporter les aspects temporelles des systèmes complexes.
   La sémantique dénotationnelle la plus populaire est celle du projet "*Stairs*" qui a fait l'objet de beaucoup de travaux et qui a été construite de façon incrémentale visant à satisfaire plusieurs objectifs [Øy03], [Øy05]. Les travaux de [Rag07] résument la sémantique dénotationnelle des *Stairs* et ses extensions.
   Dans les travaux de [Øy03], les auteurs ont proposé une sémantique de traces dénotationnelle dans laquelle ils ont défini le raffinement des diagrammes de séquence d'UML2.X. Ils ont proposé d'abord, une nouvelle catégorisation des traces, qui est différente de celle proposée par le standard. En effet, ils ont défini trois types de traces : traces valides, invalides et non-conclusives. Ensuite, ils ont défini un nouvel opérateur XALT qui capture les comportements obligatoires qui doivent être préservés quand le DS est raffiné. Enfin ils ont défini les différents types de raffinement pour les DS.

   Dans [Øy05], cette sémantique a été étendue pour inclure les contraintes temporelles. Dans [Rag05], les auteurs ont proposé une sémantique dans laquelle ils définissent de nouveaux opérateurs REFUSE et VETO qui remplacent l'opérateur NEG. Dans les travaux de [Lu 11], les auteurs ont également défini une sémantique de trace dénotationnelle, qui permet de distinguer entre les comportements requis (obligatoires) et possibles. Dans la sémantique proposée, les auteurs ont étendu la définition standard de la trace en y injectant la condition de garde afin de distinguer entre les comportements requis et possibles.
   La sémantique définie est utilisée pour la formalisation et la vérification de la relation de raffinement entre les diagrammes de séquence dans un premier temps, puis pour la vérification de la relation de conformité (*conformance*) (qui est plus générique que la relation de raffinement) entre les diagrammes de séquence.
   Dans leurs travaux préliminaires [Mar04], les auteurs ont proposé une sémantique de traces pour les diagrammes de séquence d'UML2.X, dans un style dénotationnel, qui est basée sur les structures des ensembles multiples partiellement ordonnées (*pomsets frameworks*). La sémantique proposée capture les opérateurs de composition à partir des dia-

grammes de séquences de messages hiérarchiques (*high message sequence charts (HMSC)*) et les opérateurs NEG et ASSERT.
Dans les travaux de [Yo06], les auteurs ont proposé une sémantique dénotationnelle qui est basée sur les ordres partiels "*Branching time semantics*". Les auteurs ont attribué à chaque fragment un graphe contenant les événements et leurs relations et ils ont traité les contraintes temporelles,

2. les sémantiques opérationnelles :
   Dans les travaux de [M. 06], les auteurs ont proposé une sémantique opérationnelle, qui est basée sur la sémantique dénotationnelle de *Stairs*, dans laquelle ils ont défini des méta-stratégies aussi bien pour guider et pour formaliser les comportements obligatoires, possibles et négatifs. Les comportements obligatoires sont modélisés avec un nouvel opérateur XALT qu'ils ont défini, l'opérateur ALT modélise selon leur approche les comportements possibles. L'opérateur NEG est un choix entre deux alternatives tels qu'une branche est vide et l'autre est négative.
   La sémantique opérationnelle est définie comme la combinaison de deux systèmes de transitions appelés respectivement système d'exécution et système de projection. Les deux systèmes fonctionnent en coopération : le système d'exécution met à jour le système de projection en indiquant l'état courant du medium de communication. Le système de projection met à jour le système d'exécution en sélectionnant l'événement à exécuter et en retournant l'état du diagramme après l'exécution de l'événement.
   Dans [M. 06], les auteurs ont proposé l'implémentation de la sémantique des *Stairs* en *Maude*[3].
   Dans les travaux de [Mar05], les auteurs ont proposé une sémantique opérationnelle qui est similaire à la sémantique de [M. 06] avec un traitement différent des comportements négatifs. La sémantique opérationnelle proposée est basée sur la sémantique dénotationnelle qui a été proposée dans les travaux de [Mar04]. La sémantique permet de définir et de prouver mathématiquement l'implémentation et le raffinement.
   Dans les travaux de [Nab15], les auteurs ont défini une sémantique opérationnelle pour les DS qui modélisent les comportements des systèmes réactifs. Ces systèmes peuvent produire des mots infinis. Cette caractéristique de ces systèmes a justifié leurs choix de définir une sémantique opérationnelle sous la forme d'automates de Büchi en utilisant les modèles de transformation pour la description des interactions dans les DS. Les auteurs ont considéré uniquement un sous-ensemble des fragments combinés : SEQ, STRICT, ALT, OPT, et LOOP. Par conséquent ils n'ont traité que les comportements positifs des DS,

3. les sémantiques transformationnelles :
   dans [Dav08], les auteurs ont proposé une sémantique modale pour les diagrammes de séquence d'UML2.X pour remédier aux ambiguïtés qui sont liées à l'utilisation des deux fragments combinés ASSERT et NEG qui modélisent respectivement les comportements requis et invalides.
   Dans [R. 05], les auteurs ont proposé une sémantique pour les diagrammes de séquence d'UML2.X qui est basée sur la traduction des DS en automates de Büchi, dans le but de la vérification des propriétés de sûreté et de vivacité pour les système réactifs. Ils ont utilisé la notion classique de la relation de raffinement comme une inclusion de traces pour la vérification de la relation de raffinement entre les DS. Dans [Nab15], les auteurs ont considéré des diagrammes de séquence d'UML2.X qui modélisent les comportements des systèmes réactifs. Les interactions dans de tels systèmes peuvent mener à des états infinis. Ainsi, pour la vérification formelle des diagrammes de séquence, ces derniers sont transformés vers des automates de Büchi abstraits en utilisant des transformations algé-

3. Maude est un langage très performant et un système supportant les spécifications logique et la programmation pour un grand nombre d'applications

briques.
Dans les travaux de [Da07], les auteurs ont proposé une sémantique informelle pour interpréter les DS avec des processus légers *thread tag*. Les auteurs se sont basés sur une analyse des différences entre les diagrammes basiques de séquence de messages (*basic message sequence charts (bMSC)*) et les DS ; ils ont dégagé les insuffisances de la sémantique traditionnelle pour l'interprétation des DS. La sémantique proposée capture l'aspect concurrentiel lorsque plusieurs *thread-tag* participent à l'interaction dans un DS. Les fragments combinés n'ont pas été évoqués dans leurs travaux.
Dans [C. 05a], les auteurs ont proposé une sémantique compositionnelle pour les DS en utilisant les réseaux de Pétri qui permettent de capturer la structure d'ordre partielle des DS.
Dans les travaux de [She13], les auteurs ont défini une sémantique pour les diagrammes de séquence équipés de fragments combinés imbriqués avec la logique temporelle linéaire (*linear temporal logic (LTL)*). Les auteurs ont visé la vérification de la cohérence entre un ensemble de diagrammes de séquence ainsi que la vérification des propriétés de sûreté et de non-blocage. Ils ont généré des propriétés de cohérence et de sûreté à partir de diagrammes de séquence équipés respectivement avec des fragments combinés NEG et ASSERT. L'approche a été évaluée avec le vérificateur des modèles (*model checker*) "*NuSMV*" et ils ont développé un outil pour l'implémentation,

4. la sémantique axiomatique :
   il y a peu de travaux qui ont proposé des sémantiques axiomatiques pour les diagrammes de séquence. Dans les travaux de [Kev00], les auteurs définissent une sémantique axiomatique structurée pour les modèles UML. La sémantique proposée vise à fournir une interprétation formelle de la plupart des éléments des notations d'UML permettant des analyses précises des modèles UML et leurs vérifications l'un vis à vis l'autre. Ils ont fourni pour les DS une interprétation logique structurée dans laquelle les objets communicants des DS sont représentés en utilisant la théorie des morphismes. Ils ont motivé leurs choix d'une sémantique axiomatique par le fait que ce type de sémantique permet d'établir une relation directe entre les outils de preuves pour les notations UML et la sémantique,

5. la sémantique co-algébrique :
   dans les travaux [Kev09], les auteurs ont défini une sémantique co-algébrique pour un développement basé sur les composants en UML ; la sémantique a été proposée pour l'appliquer dans le contexte de composition et de restructuration (*refactoring*) des diagrammes de séquence.

2.3.4 Critiques de l'existant

Bien que la sémantique standard ait fourni une syntaxe précise pour les diagrammes de séquence, les définitions qui permettent de calculer l'ordre partiel entre les événements ne sont pas formalisées, ce qui complique l'interprétation et l'analyse des DS. Ces ambiguïtés intentionnelles de la sémantique standard, appelées points de variations, mènent à plusieurs choix subtiles dans l'interprétation du langage. Même les DS basiques sont interprétés différemment par les approches existantes [Zol11].
Les fragments combinés permettent la modélisation des comportements complexes des systèmes. Cependant, ils compliquent davantage l'interprétation de diagrammes de séquences et ils présentent des défis pour le calcul de l'ordre partiel entre les événements. Pour surmonter les difficultés et contourner les problèmes ou les inconsistances émanant de la présence des FC dans un DS, quelques approches émettent des hypothèses assez strictes et ne traitent pas convenablement quelques CF (ALT, LOOP).

Dans les travaux de [C. 05a], les auteurs utilisent une sémantique boîte noire (*black box semantics*), qui consiste à abstraire le FC et le considérer comme un événement atomique de telle façon que le FC n'interfère pas avec les comportements qui sont à l'extérieur du FC. Ceci correspond à un séquencement strict entre les événements avant et après le FC ; l'entrée au FC est faite de façon synchrone par toutes les lignes de vie.
Pour les autres travaux existants, ils suivent deux approches différentes avant d'effectuer n'importe quel traitement sur les diagrammes de séquence tels que :
• la première catégorie des travaux [Mar04], [Ale06], [Øy03], [Øy05], [Har03], [C. 05a], [Yo06], [Da07], [JR07], [Hui08] procèdent à la décomposition du DS avec FC puis à l'éclatement des FC (par exemple les branches d'un fragment combiné ALT sont divisées et les itérations d'un fragment combiné LOOP sont supprimées (voir illustration 1 et 2). En procédant ainsi des DS basiques sont obtenus et les traitement sur ces diagrammes ne sont pas compliqués.

Illustration : l'ensemble des traces qui sont associées au diagramme de séquence, représenté par la Figure 2.11, est l'union des ensembles des traces des diagrammes de séquence représentés par la Figure 2.12 :

$$\{\ !m1, ?m1\ \} \bigcup \left\{ \begin{array}{l} !m1, ?m1, !m2, ?m2, !m3, ?m3 \\ !m1, ?m1, !m2, !m3, ?m2, ?m3 \\ !m1, ?m1, !m2, !m3, ?m3, ?m2 \end{array} \right\} \bigcup \left\{ \begin{array}{l} !m1, ?m1, !m4, ?m4 \\ !m1, !m4, ?m1, ?m4 \\ !m1, !m4, ?m4, ?m1 \end{array} \right\}$$

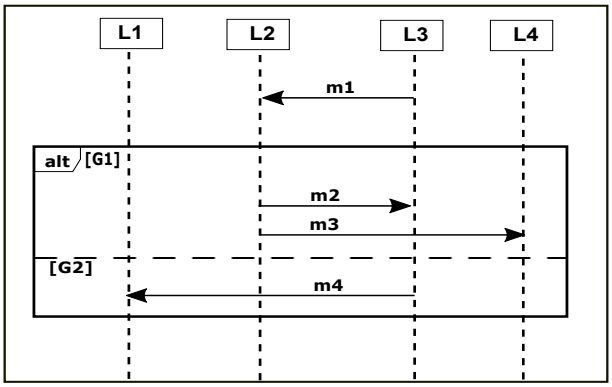

FIGURE 2.11 – Diagramme de séquence avec FC ALT

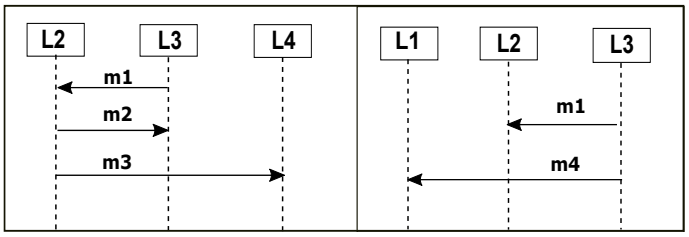

FIGURE 2.12 – Aplatissement du DS de la Figure 2.11

Illustration : les traces associées au DS de la Figure 2.13 sont toutes les traces qui respectent les contraintes de précédence suivantes entre les événements : sur la ligne de vie L2 nous avons $\{!m1^1 <!m2^1, !m2^1 <!m1^2, !m1^2 <!m2^2\}$ [4], sur la ligne de vie L1 nous avons $\{?m1^1 <?m1^2\}$ et sur la ligne de vie L3 nous avons $\{?m2^1 <?m2^2\}$.

Pour le calcul de chaque trace d'un DS avec FC, la sémantique standard recommande de calculer pour chaque composant du DS (interactions basiques, FC) sa trace, puis de les assembler en les combinant avec l'opérateur faible séquencement, ce qui revient à la même procédure

4. mi^j : i désigne le numéro du message et j désigne le numéro de l'itération

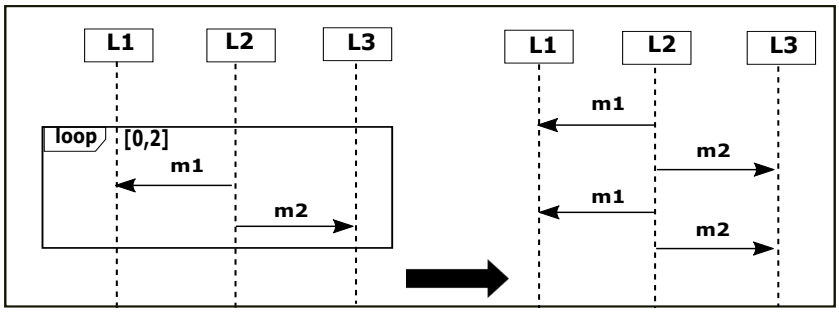

FIGURE 2.13 – Aplatissement d'un diagramme de séquence avec FC LOOP

de décomposition. Notons que généralement la représentation intermédiaire des DS n'est pas fournie par ces approches. Ce prétraitement des diagrammes de séquence fait perdre l'intérêt de la représentation graphique compacte offerte par les FC. D'autant plus que nous pouvons pas perdre des informations de coordination quand on extrait des vues partielles du DS.

En outre, pour un diagramme de séquence basique, la sémantique standard [Obj15] interprète l'ordre des événements sur chaque ligne de vie en lisant le DS du haut vers le bas. Dans le contexte d'un système distribué, les objets sont indépendants, la communication est asynchrone et dans le modèle général de communication, les réceptions des messages ne sont pas ordonnées. Dans certains cas particuliers, par exemple l'existence d'une file d'attente (*first in first out*) *FIFO* sur la ligne de vie réceptrice ou lorsque les messages sont envoyés par une même ligne de vie, les réceptions des messages sont ordonnées. Par conséquent, imposer un ordre total des événements sur chaque ligne de vie, pour des DS qui modélisent les comportements de tels systèmes, mènent à la perte de quelques comportements possibles, ce qui cause l'émergence de comportements non-spécifiés durant l'implémentation et la violation de propriétés de sûreté.
La contrainte de l'ordre total sur chaque ligne de vie, peut être relâchée par l'utilisation de l'opérateur COREGION, qui appliqué sur une ligne de vie permet un parallélisme implicite [5].
Cependant, dans quelques cas particuliers (voir Figure 2.14), l'ajout du fragment combiné COREGION uniquement ne résout pas le problème de l'ordre total sur une ligne de vie ; des messages additionnels doivent être ajoutés pour rétablir certaines relations de précédences entre quelques événements [Sim04].

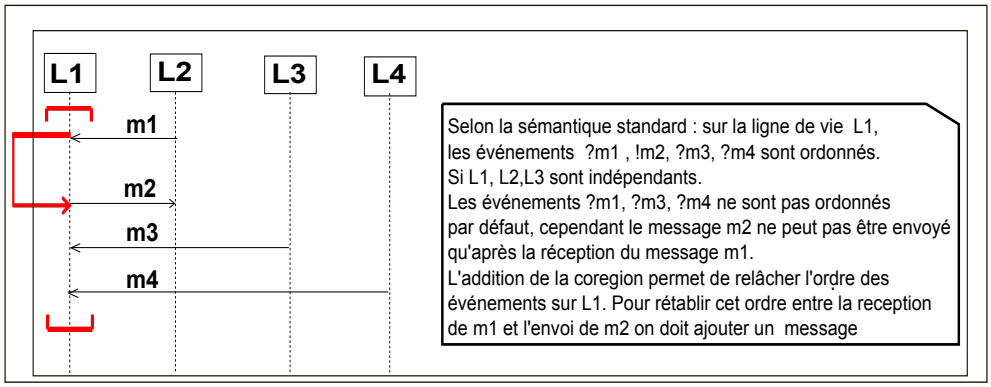

FIGURE 2.14 – Diagramme de séquence basique avec ajout de FC COREGION et message additionnel

De plus, lorsque nous avons un DS qui contient plusieurs fragments combinés imbriqués, nous aurons une représentation graphique surchargée qui est due au chevauchement entre les

5. i.e.les événements couverts par l'opérateur COREGION peuvent se produire dans n'importe quel ordre

FC existants et les opérateurs COREGION rajoutés, ce qui peut causer des difficultés pour la délimitation du début et de la fin de chaque opérateur COREGION.

• la seconde catégorie des travaux [Ale04], [Jul06], [Dav08] traitent le DS comme un tout. Les travaux qui suivent cette approche empruntent le concept de localisations (*locations concept*), qui est emprunté du formalisme *Live Sequence Charts (LSC)*, pour marquer l'occurrence des événements. Ils encodent l'ordre partiel entre les événements du DS en des structures finies (comme les automates [Zol11], les automates de Büchi [R. 05], [Sha11], les systèmes de transitions hiérarchiques [Hui08], les automates communicants [Hes06], les graphes [Yo06]). Cependant, ce traitement entraîne une explosion exponentielle de la représentation [Ale14]. D'autant plus que dans ces approches les auteurs émettent quelques restrictions syntaxiques (par exemple dans l'approche de [Sha11], ils considèrent uniquement des messages synchrones).
Le Tableau 2.1 récapitule la classification des différentes approches selon le prétraitement effectué sur les diagrammes de séquence d'UML2.X (soit l'aplatissement du diagramme de séquence ou comme un tout).

Maniement du diagramme de séquence		
	Aplatissement	Totalité
Approches	[Øy05], [C. 05a], [Har03] [Ale06], [JR07] [Yo06], [Da07] [Obj15], [Mar04]	[Ale04] [Jul06] [Dav08]

TABLE 2.1 – Maniement du diagramme de séquence par les sémantiques existantes

Les approches existantes couvrent généralement quelques fragments combinés. Nous illustrons dans les Tableaux 2.2 et 2.3 les sémantiques des DS les plus connues ainsi que les aspects couverts par leurs approches.
Nous notons, même, que certaines approches redéfinissent quelques FC selon l'objectif visé. Dans l'approche de [Øy05], les auteurs introduisent l'opérateur XALT *vs* ALT, pour distinguer entre l'indéterminisme explicite, permettant de capturer le comportement obligatoire, et l'indéterminisme implicite permettant de capturer le comportement possible.
Dans l'approche de [Rag05], les auteurs définissent les opérateurs REFUSE et VETO au lieu de l'opérateur NEG pour surmonter les ambiguïtés d'interprétations de cet opérateur.

La sémantique standard recommande l'application de l'opérateur faible séquencement pour combiner les traces de différents constituants d'un DS avec FC, ce qui engendre un chevauchement de l'occurrence des événements et mène à la génération de traces non-intuitives. Ce problème est mieux explicité dans le cas d'un fragment combiné LOOP, où nous observons un chevauchement de l'occurrence des événements qui appartiennent à différentes itérations. Ceci n'est pas intuitif puisqu'il ne correspond pas au sens ordinaire du fragment combiné LOOP.
Le fragment combiné ALT, qui modélise des comportements alternatifs, a une signification plus générale que la structure conditionnelle SI-ALORS-SINON. Cependant, pour éviter ces ambiguïtés et les cas pathologiques (comme le mentionne l'auteur dans ces travaux [Yo06]) qui peuvent résulter de l'utilisation de ces FC, certaines approches adoptent plutôt une signification similaire à celle des langages de programmation structurés. Ainsi dans le fragment combiné LOOP l'opérateur STRICT SEQUENCING est appliqué ([Jul06], [Yo06]) et le fragment combiné ALT est considéré équivalent à la structure conditionnelle *Si-Alors-Sinon*. Cependant, ces restrictions limitent le pouvoir d'expression de ces opérateurs et fait perdre quelques traces possibles.

Illustration : considérons le DS représenté par la Figure 2.13, les traces obtenues en interprétant le fragment combiné LOOP de façon similaire à celle dans les langages de programmation structurée, (nous avons un séquencement strict entre les itérations par conséquent tous les évé-

nements de la première itération se produisent avant d'exécuter la deuxième itération), sont les suivantes :

$$\left\{\begin{array}{l} !m1^1, !m2^1, ?m2^1, ?m1^1, !m1^2, !m2^2, ?m2^2, ?m1^2, \\ !m1^1, !m2^1, ?m2^1, ?m1^1, !m1^2, !m2^2, ?m1^2, ?m2^2, \\ !m1^1, !m2^1, ?m2^1, ?m1^1, !m1^2, ?m1^2, !m2^2, ?m2^2, \\ !m1^1, !m2^1, ?m1^1, ?m2^1, !m1^2, !m2^2, ?m2^2, ?m1^2, \\ !m1^1, !m2^1, ?m1^1, ?m2^1, !m1^2, !m2^2, ?m1^2, ?m2^2, \\ !m1^1, !m2^1, ?m1^1, ?m2^1, !m1^2, ?m1^2, !m2^2, ?m2^2, \\ !m1^1, ?m1^1, !m2^1, ?m2^1, !m1^2, !m2^2, ?m2^2, ?m1^2, \\ !m1^1, ?m1^1, !m2^1, ?m2^1, !m1^2, !m2^2, ?m1^2, ?m2^2, \\ !m1^1, ?m1^1, !m2^1, ?m2^1, !m1^2, ?m1^2, !m2^2, ?m2^2 \end{array}\right\}$$

Approches / Aspects traités	[Har03]	[Øy05], [Hal07], [Rag05]	[Ale04]	[Da07]	[Lu 11]
Garde		*	*		*
ALT	*	*	*	*	*
OPT	*	*		*	*
LOOP	*	*		*	*
BREAK	*				
PAR	*	*	*	*	*
SEQ	*	*	*	*	*
STRICT	*	*	*	*	
CRITICAL	*				*
NEG	*	*	*		
ASSERT	*	*	*		
IGNORE	*				
CONSIDER	*				

Table 2.2 – Fragments combinés traités par les travaux existants

Approches / Aspects traités	[Mar05]	[Jul06]	[R. 05]	[Yo06]	[Ale06]	[She13]	[Gab08]
Garde		*		*	*		
ALT	*	*	*	*	*	*	*
OPT	*			*	*	*	
LOOP	*		*	*	*	*	
BREAK				*			
PAR	*	*		*	*		*
SEQ	*	*	*	*			
STRICT	*			*	*	*	
CRITICAL						*	
NEG	*	*	*		*		*
ASSERT	*	*					*
IGNORE	*				*	*	*
CONSIDER	*					*	*

Table 2.3 – Fragments combinés traités par les travaux existants (suite)

2.3.4.1 Problèmes liés aux fragments combinés comportant une garde.

Les gardes des FC conditionnent l'occurrence des événements qui sont couverts par le FC, de ce fait elles impactent le calcul des traces et ne doivent pas être ignorées.

Illustration : nous montrons à travers cette illustration comment la considération de la garde influe sur le nombre de traces qu'on peut générer à partir d'un DS. Considérons le DS représenté par la Figure 2.15. En ignorant la garde (c'est à dire la garde est supposée être toujours vraie), nous obtenons une seule trace qui est $\{(!m1, ?m1)\}$. En considérant la garde nous obtenons trois traces possibles $\{\varnothing, (!m1, ?m1), (!m1, ?m1, !m1, ?m1)\}$, en effet la garde peut être fausse dès le début donc aucun événement n'est exécuté, la garde peut devenir fausse après une seule itération ou la garde peut être toujours vraie.

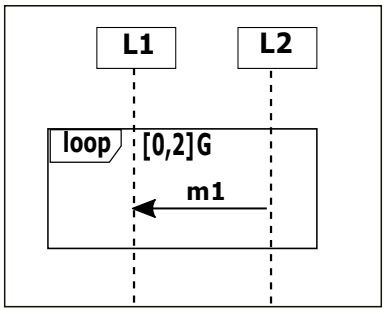

FIGURE 2.15 – Diagramme de séquence avec FC LOOP comprenant une garde

Il y a des ambiguïtés importantes liées aux gardes ; elles concernent, principalement, leurs méthodes d'évaluation. En effet, plusieurs aspects doivent être pris en considération ; il s'agit de déterminer qui doit évaluer la garde, quand, comment et combien de fois cette évaluation doit être faite.

Les gardes peuvent être évaluées soit par chaque ligne de vie [Hui08], ou par toutes les lignes de vie couvertes par le FC [Yo06]. Dans la littérature, le temps d'évaluation des gardes est traité de deux manières [Zol11] :

1. soit de façon implicite : ceci concerne les sémantiques compositionnelles [Har03], [Øy05], [Mar04], [Hai07],
2. soit de façon explicite [C. 05a], [Yo06], [R. 05], [Sha11], [Ale06], [JR07], [Ale04], [Jul06], [Hui08]. Cependant dans ces travaux des hypothèses stricts sont émises pour prévenir les inconsistances qui causent des problèmes de synchronisation.
3. dans la première catégorie ([C. 05a], [Yo06], [R. 05], [Sha11], [Ale06], [JR07]), les travaux qui imposent que les lignes de vie qui sont couvertes par le FC doivent synchroniser avant d'effectuer le choix de l'opérande et un seul choix global est fait.
4. dans la seconde catégorie [Ale04], [Jul06], [Hui08], l'approche proposés semble être adéquate pour un contexte de composants distribués, puisque chaque ligne de vie peut faire son choix de façon indépendante, cependant ils imposent que l'évaluation des gardes des opérandes s'effectuent dans l'ordre d'apparition des opérandes.

Un autre problème qui doit être géré, qui découle de l'évaluation de garde, est la synchronisation entre les lignes de vie le cas des fragments combinés ALT et LOOP gardés. En effet, la sémantique standard impose pour un fragment combiné ALT, qu'un seul opérande doit être exécuté parmi plusieurs opérandes potentiels. Si aucune garde n'est vraie, le FC est omis. Dans le contexte de DS modélisant un comportement de système distribué, puisque les lignes de vie impliquées dans l'interaction sont indépendantes, un problème de coordination surgit. Par conséquent, dans un contexte de composants distribués, une fois que la garde est évaluée, la décision doit être propagée correctement aux autres lignes de vie qui sont couvertes par le FC pour prévenir une exécution en parallèle de plusieurs opérandes et éviter l'émergence de comportements

	Explicite		Implicite
Méthode	Choix global	[C. 05a], [Yo06], [R. 05], [Sha11], [Ale06], [JR07]	[Har03], [Øy05], [Mar04], [Hai07]
	Choix indépendant	[Ale04], [Jul06], [Hui08]	

TABLE 2.4 – Classification des approches des sémantiques existantes pour le traitement des problèmes liés aux gardes

non spécifiés durant l'implémentation. De plus, pour un opérande LOOP gardé, les événements, qui sont à l'intérieur de l'opérande, sont exécutés tant que la garde est évaluée positivement et le nombre maximal d'itérations n'est pas encore atteint. Comme nous l'avons déjà expliqué pour le fragment combiné LOOP les traces qui contiennent un chevauchement entre l'occurrence des événements des différentes itérations sont possibles. Donc, une fois la garde devient fausse, les événements des itérations précédentes, qui ne se sont pas encore déclenchés, doivent se déclencher. Quant aux événements des itérations restantes, ils ne doivent pas se déclencher.

Dans le cas d'un fragment combiné STRICT, la synchronisation entre les opérandes est aussi primordiale, puisque les opérandes sont exécutés successivement et les événements d'un opérande intermédiaire ne peuvent se déclencher que si les événements de l'opérande précédent sont consommés.

Dans la sémantique standard, l'ambiguïté réside dans le fait qu'elle est compositionnelle, donc ceci suppose que l'évaluation de la garde est implicite, bien que dans la version la plus récente de l'OMG, il y a une indication que la garde doit être placée, sur la ligne de vie où se déclenche le premier événement, juste avant lui. Cependant, le premier événement d'un opérande n'est pas bien défini.
Le Tableau 2.4 récapitule la classification des approches des sémantiques existantes pour le traitement des problèmes liés aux gardes.

2.3.4.2 Problèmes liés aux fragments combinés imbriqués.

Les FC peuvent être imbriqués pour permettre de modéliser des scénarios sophistiqués, qui peuvent être agrégés dans un diagramme de séquence unique. Par exemple imbriquer des alternatives dans un fragment combiné LOOP permet d'exprimer des différents choix plusieurs fois. Le contraire permet de modéliser des actions itératives de certaines alternatives choisies.

Dans la sémantique standard et les sémantiques compositionnelles [Obj15], [Yo06], [Har03], [Øy05], [Mar04], [Hai07], les fragments combinés imbriqués sont implicitement supportés et la profondeur d'imbrication des FC n'est pas limitée. Bien qu'en se focalisant sur l'imbrication de certains types de FC beaucoup de problèmes peuvent apparaître. En effet, l'imbrication de certains FC peut ne pas être significative ou peut engendrer des ambiguïtés d'interprétations, d'autres imbrications de certains FC ne sont pas possibles [Gab08]. Dans les travaux de [Gab08], les auteurs étudient les problèmes possibles qui résultent de l'imbrication des opérateurs de conformité (ASSERT, IGNORE, CONSIDER, NEG), l'opérateur ALT et OPT.

Les travaux qui ont traité explicitement les FC imbriqués ont émis plusieurs restrictions syntaxiques. Par exemple dans les travaux de [Gab08], [Jul06], les auteurs ont considéré l'imbrication d'un nombre réduit de FC en limitant le niveau d'imbrication à deux. Dans les travaux de [Gab08], les auteurs ont considéré uniquement les opérateurs de conformité (ASSERT, IGNORE, CONSIDER et NEGATE) permettant de catégoriser les traces en des traces valides et invalides, ainsi que les fragments combinés ALT et PAR. Ils ont visé dans leurs travaux, de définir des lois qui autorisent ou interdisent l'imbrication de ces opérateurs en limitant le niveau d'imbrication à deux. Dans les travaux de [R. 05], les auteurs ont traité uniquement les combinaisons des DS

basiques sans les fragments combinés. Dans les travaux de [Ale06], les auteurs autorisent uniquement l'imbrication des fragments basiques dans des fragments combinés NEG.
Dans les travaux de [She13], les auteurs ont proposé une sémantique formelle pour les DS en tenant compte des FC imbriqués. La sémantique proposée est très liée au formalisme cible qui est la logique temporelle linéaire (*linear temporal logic* (*LTL*)) et les formalisations proposées sont assez complexes. De plus, ils ont considéré les diagrammes de séquence d'UML2.X qui ont des traces finies tandis que le formalisme LTL représente, à la base, des traces infinies.

2.4 Conclusion

Dans ce chapitre nous avons présenté les concepts de base des diagrammes de séquence.
Nous avons également donné un état de l'art sur les travaux existants qui ont proposé une panoplie de sémantiques visant différents objectifs.
Nous avons souligné les insuffisances de la sémantique standard ainsi que de celles des sémantiques existantes en se focalisant sur les problèmes qui nous intéressent au sein de nos travaux.
Dans la suite de notre travail, nous allons tâcher d'apporter des solutions à ces insuffisances en commençant par la proposition d'une sémantique plus appropriée aux diagrammes de séquence modélisant les comportements des systèmes distribués. En outre, nous visons aussi qu'avec la sémantique proposée les DS sont interprétés de façon non ambiguë.

Chapitre 3

La sémantique causale des fragments combinés et ses extensions

Sommaire

3.1 Introduction

Bien que les diagrammes de séquence (DS) soient les prédécesseurs des diagrammes de séquence de messages (MSC), l'OMG a donné des définitions assez restrictives pour le calcul de l'ordre entre les occurrences des événements d'un DS basique. Cet ordre ne permet pas le calcul de toutes les traces possibles pour certains types de systèmes, spécifiquement pour les systèmes

distribués. Cependant l'ordre des événements dans les MSC peut être sujet à plusieurs interprétations sémantiques. En effet, dans les travaux de [Raj96], les auteurs préconisent qu'il existe plusieurs types d'ordres partiels entre les événements d'un MSC qui dépendent étroitement soit de l'architecture considérée soit de l'existence d'un intermédiaire de communication ou de files d'attentes (une ou plusieurs FIFO pour tous les processus [6]).
Les différents ordres partiels définis sont :

• la relation d'*ordre visuel* qui correspond à l'ordre d'apparition des événements dans le diagramme mais qui ne reflète pas l'ordre d'occurrence des événements pendant l'exécution du système ; ce problème est connu sous le nom *race condition*. Par exemple pour deux messages qui sont envoyés par deux processus différents vers un même processus, leurs réceptions peuvent être dans le même ordre que leurs émissions comme elles peuvent être dans l'ordre inverse,
• la relation d'*ordre forcé* qui contient toutes les paires d'événements dont l'architecture considérée garantit les occurrences dans un ordre bien déterminé, par exemple s'il existe une FIFO pour chaque processus, les réceptions de deux messages exactement dans l'ordre de leurs émissions sont garanties,
• la relation d'*ordre déduit* qui est définie comme la fermeture transitive de l'ordre forcé.

3.2 Vers le choix d'une sémantique dédiée aux DS modélisant les comportements des systèmes distribués

Dans les travaux de [Chr06], les auteurs ont proposé une sémantique nommée *sémantique causale* qui est dédiée aux DS modélisant les comportements d'un système distribué. Elle est basée sur la théorie d'ordre partiel ; ses lois qui régissent la définition de l'ordre entre les événements tiennent compte de l'indépendance des composants impliqués dans l'interaction.

Pour mettre en avant les avantages de la sémantique causale, les auteurs se sont basés sur l'inadéquation des définitions de la sémantique standard d'UML pour les systèmes distribués et ils ont présenté trois types de sémantique opérationnelle des diagrammes de séquence : la sémantique *linéaire*, la sémantique d'*émission* et la sémantique des *MSC* qui peuvent être utilisées pour la modélisation des systèmes de différents types.
Les deux premières sémantiques sont appropriées aux diagrammes de séquence qui modélisent des applications interactives ; la sémantique des *MSC* est la base de la sémantique d'UML.

La sémantique linéaire [Ja01] permet d'ordonner tous les messages du diagramme en imposant une sérialisation totale des événements associés à tous les messages. Ainsi, un message ne peut être n'est émis que si le message qui le précède a été bien reçu, même si les événements associés aux messages appartiennent à des lignes de vie indépendantes. Cette sémantique n'est applicable que pour les diagrammes de séquence séquentiels.
La sémantique d'émission [Jan02] impose une sérialisation des événements d'envoi de tous les messages du diagramme.
Les lois de ces sémantiques sont très restrictives et ne sont réalisables que pour des diagrammes de séquence modélisant des systèmes ayant des spécificités très particulières.

Illustration : pour illustrer les relations d'ordres de ces sémantiques, nous considérons le DS basique représenté par la Figure 3.1. Nous représentons les relations de précédence par des graphes de dépendance en appliquant les lois de la sémantique considérée. Les nœuds du graphe représentent les événements d'envoi ou de réception qui sont associés aux messages du diagramme et les arcs représentent les relations de précédence. Ainsi les Figures 3.2, 3.3 et 3.4 représentent les graphes de dépendances qui sont obtenus en appliquant respectivement les lois de la sémantique linéaire, de la sémantique d'émission et de la sémantique standard.

6. un processus dans un MSC est équivalent à une ligne de vie dans un DS

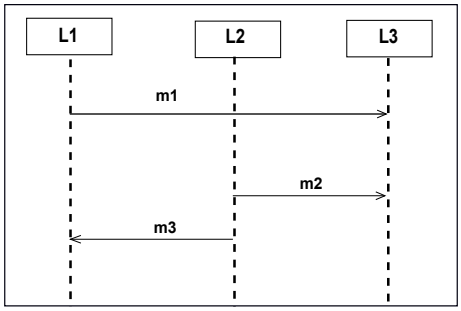

FIGURE 3.1 – Diagramme de séquence

3.2.1 Principe de la sémantique causale

La sémantique causale relâche la contrainte d'ordonnancement total des événements sur chaque ligne de vie qui est imposée dans les définitions de la sémantique standard. En effet la relation d'ordre définie est basée sur les causalités et tient compte de la relation de précédence entre les événements, ainsi que sur la synchronisation entre les lignes de vie, qui est assurée par les événements échangés sans l'utilisation d'un contrôleur global ou l'ajout de messages. Intuitivement, les lois de la sémantique causale [C. 05b] sont basées sur l'idée de n'ordonner les événements d'un diagramme de séquence que s'il y a une raison logique de le faire.
En effet, sur une même ligne de vie, deux événements d'émission et qui sont successifs sont nécessairement ordonnés. Cependant, sur une même ligne de vie, la réception de deux événements qui sont envoyés par deux lignes de vie indépendantes ne sont pas nécessairement ordonnés ; deux événements qui appartiennent à une même ligne de vie ne sont ordonnés que si l'un est la conséquence directe ou indirecte de l'occurrence de l'autre (Figure 3.6).

Dans ce qui suit, nous présentons les lois de la sémantique causale telles qu'elles ont été définies dans les travaux de [O. 05].
La relation de Synchronisation $<_{Sync}$. Cette relation permet de synchroniser le comportement de chaque ligne de vie ; chaque message m ne peut être événement de réception que s'il a été envoyé au préalable.
La relation de Réception-Émission$<_{RE}$. La réception d'un message cause l'envoi du message qui lui est consécutif.
La relation d'Émission-Émission $<_{EE}$. Si deux messages sont envoyés par la même ligne de vie, leurs émissions respectives sont ordonnées.
La relation de causalité $<_{caus}$. Cette relation est définie comme une fermeture transitive sur l'union des relations susmentionnées.

$$<_{caus}=<_{sync} \bigcup <_{RE} \bigcup <_{EE}$$

La fermeture transitive de la relation $<_{caus}$, qui est notée $<^{+}_{caus}$ permet d'obtenir toutes les

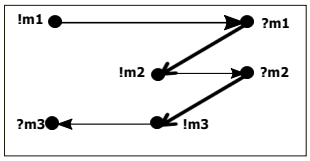

FIGURE 3.2 – Graphe de dépendance : sémantique linéaire

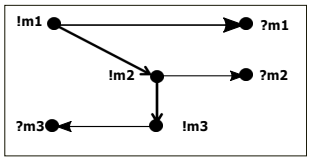

FIGURE 3.3 – Graphe de dépendance : sémantique d'émission

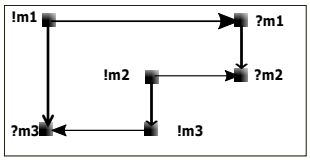

FIGURE 3.4 – Graphe de dépendance : sémantique standard

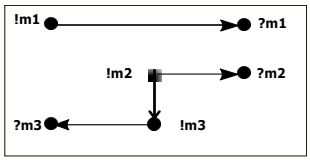

FIGURE 3.5 – Graphe de dépendance : sémantique causale

dépendances causales ou les relations de précédence entre les événements du DS.

Illustration : la Figure 3.5 représente le graphe de dépendance obtenu en appliquant les lois de la sémantique linéaire au DS représenté par la Figure 3.1.

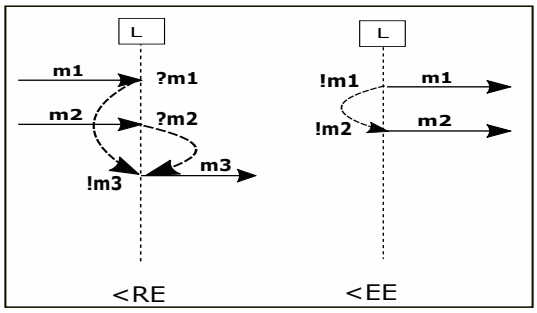

FIGURE 3.6 – Les relations de précédence $<_{RE}$ et $<_{EE}$

La sémantique causale affaiblit la sémantique standard d'UML en supprimant quelques relations de précédence entre les événements du DS considéré. D'où nous obtenons un diagramme de séquence plus expressif, qui décrit un nombre plus important de comportements acceptables, ainsi nous évitons les comportements *non spécifiés* ou *non voulus* [Raj96] [Seb01] pendant l'implémentation.

3.2.2 Insuffisances de la sémantique causale de base

La sémantique causale avait été proposée uniquement pour les diagrammes de séquence basiques. Si nous tentons d'appliquer ses lois pour les DS avec des fragments combinés on reporte quelques problèmes de génération des relations aberrantes qui causent des blocages de certains événements.

Illustration : considérons le diagramme de séquence, comportant un fragment combiné ALT, représenté par la Figure 3.7. Le graphe de dépendance relatif à ce diagramme est représenté par la Figure 3.8. Selon la relation $<_{EE}$, sur la ligne de vie $L1$, nous avons l'événement $!m1$ précède l'événement $!m2$ qui est une relation aberrante qui cause le blocage de l'événement $!m3$.

Dans ce qui suit nous proposons l'extension des lois de la sémantique causale pour traiter les diagrammes de séquence d'UML2.X qui sont dotés de structures de haut niveaux : les fragments combinés.

3.3 Première extension de la sémantique causale

Comme nous l'avons mentionné dans le chapitre1, la présence des FC dans les DS complique certaines tâches liées à la détermination des relations de précédence de chaque événement, sachant qu'il faut prendre en compte les spécificités de la sémantique de chaque FC sans oublier

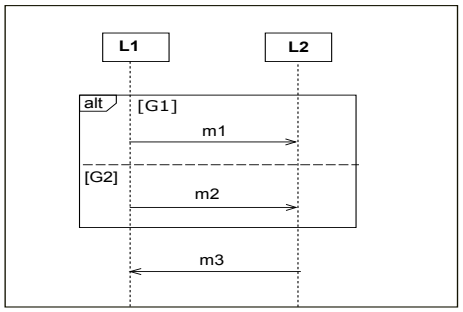

FIGURE 3.7 – DS comportant un FC ALT

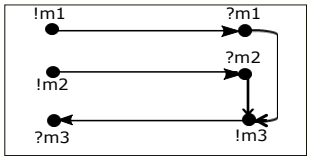

FIGURE 3.8 – Graphe de dépendance associé au DS de la Figure 3.7

les problèmes liées aux gardes des FC dont la considération induit plusieurs problèmes de synchronisation entre les lignes de vie. Nous avons également montré que les approches existantes [Obj09], [Yo06], [Ale06], [C. 05a], [Mar05], [Øy05], [Har03] restreignent l'utilisation des FC en émettant des hypothèses assez restrictives ou en redéfinissant carrément d'autres opérateurs pour simplifier leurs utilisations et éviter les problèmes liés à leurs présences.

Dans le but de simplifier l'approche et de la généraliser, nous avons procédé par étapes. Dans une étape préliminaire, nous avons émis quelques hypothèses assez strictes ; en effet nous avons considéré des diagrammes de séquence comportant quelques fragments combinés (ALT, OPT et LOOP) qui sont disposés de façon séquentielle ; ce travail a fait l'objet d'une première publication [Fat15]. Puis nous avons généralisé l'approche en considérant d'autres FC ainsi que leurs imbrication.
Dans ce qui suit, nous commençons par donner les définitions formelles d'un diagramme de séquence comportant des fragments combinés ALT, OPT et LOOP, qui sont disposés en séquentiel.
Dans ce qui suit, nous commençons par donner les définitions formelles d'un diagramme de séquence comportant des FC ALT, OPT et LOOP, qui sont disposés en séquentiel.

3.3.1 Formalisation des diagrammes de séquence

3.3.1.1 Bref aperçu sur les travaux formalisant les notations UML

Dans la littérature, nous retrouvons plusieurs travaux qui ont formalisé les notations d'UML en utilisant différents formalismes tels que le formalisme Z [Kha05], les processus séquentiels communicants (CSP) [Jac14] [Li 10]et les systèmes de vérification de prototypes (PVS) [B. 02]. Le formalisme Z semble inadéquat pour la formalisation des diagrammes de séquence puisqu'il ne supporte pas les aspects dynamiques et ne dispose pas d'outils de vérification assez puissants. Par contre, le langage CSP permet de modéliser les aspects dynamiques. Des approches qui visent à combiner les deux aspects peuvent être intéressantes pour la formalisation des diagrammes de séquence. Contrairement aux formalismes Z et CSP, PVS est un environnement qui offre, d'une part des outils puissants de vérification tels que l'explorateur de modèles (*model checker*) et le prouveur de théorèmes, d'autre part il supporte les notions de séquences, d'enregistrements et des listes, qui facilitent la spécification des notations des diagrammes de séquence.

Dans notre approche, nous avons choisi les notations qui sont basées sur la théorie des ensembles. C'est une alternative qui nous procure une multitude d'avantages. En effet, bien qu'elle est fondée sur la logique du premier ordre, elle permet la manipulation des objets d'ordre supérieur tels que les ensembles et les relations de n'importe quelle profondeur (c'est-à-dire des ensembles et des relations qui sont eux-mêmes construits sur des ensembles et des relations) [J.-96b].

3.3.1.2 Formalisation des diagrammes de séquence et des fragments combinés

Definition 1. ***Diagramme de séquence***
Un diagramme de séquence DS est un tuple

$$DS : \langle L, M, EVT, FCT\_s, FCT\_r, FCT\_l, F, <_{caus} \rangle$$

où :
- *L est l'ensemble des lignes de vie,*
- *M est l'ensemble des messages,*
- *EVT est l'ensemble des événements qui est composé des ensembles E_s et E_r qui représentent respectivement l'ensemble des événements d'émission et de réceptions,*
- *FCT_s et FCT_r sont deux fonctions bijectives totales qui permettent d'associer à chaque message respectivement un événement d'émission et un événement de réception,*
- *FCT_l est une fonction surjective totale qui permet d'associer à événement une ligne de vie,*
- *F est une séquence de k fragments combinés* ALT, OPT *et* LOOP,
- $<_{caus}$ *est la relation d'ordre partiel.*

Nous définissons la relation d'ordre partiel entre les événements dans chaque ligne de vie, qui est notée $<^*_{DS,l}$. La relation $<^*_{DS,l}$ est l'ensemble des paires d'événements (e, e') tels que les événements e et e' appartiennent à l'ensemble de tous les événements ; les événements e et e' sont directement consécutifs et ils sont envoyés ou événement de réceptions par la ligne de vie L. La relation locale $<^*_{DS,l}$ est une relation minimale non cyclique et non transitive.
Nous définissons également la fonction *F_op* qui est une fonction totale qui permet d'associer à chaque opérande son FC correspondant.

Definition 2 (Messages bien formés.)**.** *L'ensemble des messages M est bien formé si chaque message est identifié par une paire d'événements : un événement d'émission et un événement de réception.*

Definition 3 (Fragment combiné conditionnel ALT, OPT)**.** *Un fragment combiné conditionnel est une séquence de couples qui sont formés d'une garde et d'un opérande*

$$F = \langle (guard_1, OP_1), (guard_2, OP_2), ..., (guard_i, OP_l) \rangle$$

où : $guard_i$ est la contrainte qui garde la $i^{ème}$ opérande, OP_i est l'ensemble des événements qui sont couverts par la $i^{ème}$ opérande.

Definition 4 (Fragment combiné LOOP)**.** *Un fragment combiné* LOOP *est un quadruplet*

$$F = \langle guard, min, max, OP \rangle$$

où : guard est la contrainte qui garde l'opérande LOOP*, min est le nombre minimal d'itérations, max est le nombre maximal d'itérations et OP est l'ensemble des événements qui sont couverts par un fragment combiné* LOOP *.*

3.3.2 La relation d'ordre partiel $<_{caus}$

L'occurrence d'un événement dépend de la relation d'ordre partiel $<_{caus}$. Nous étendons cette relation en redéfinissant les lois causales pour supporter les fragments combinés ALT, OPT et LOOP.
Pour illustrer notre approche, nous considérons un exemple de diagramme de séquence comportant deux fragments combinés $F = \langle F_1, F_2 \rangle$ représenté par la Figure 3.9, où :

— $F_1 = \langle (guard_1, OP_1), (guard_2, OP_2) \rangle$ est un fragment combiné ALT qui modélise uniquement deux comportements, cependant la modélisation proposée est valide pour un fragment combiné avec k opérandes.
— $F_2 = \langle guard, 0, N, OP3 \rangle$ est le FC LOOP.

Nous considérons l'ensemble $OP = OP_1 \cup OP_2 \cup OP_3$. Nous définissons une fonction totale *F_op* qui permet d'associer à chaque opérande dans un DS donné le fragment combiné correspondant.

Relation Émission-Émission $<_{EE}$. Dans cette relation, nous énonçons les conditions nécessaires pour ordonner deux événements d'émission, qui appartiennent à la même ligne de vie et qui sont successifs.

Relation Réception-Émission $<_{RE}$. Dans cette relation, nous énonçons les conditions nécessaires pour ordonner un événement de réception et un événement d'émission, qui appartiennent à la même ligne de vie et qui sont successifs.

Nous définissons une nouvelle relation $<_{guard}$, telle que si la garde du fragment combiné dépend de l'exécution d'un événement qui précède le FC nous ajoutons une relation de causalité entre cet événement et le premier événement du FC qui est déclenchable. Notons que le premier événement d'un FC qui est autorisé à être exécuté n'est pas nécessairement le premier événement qui apparaît dans le FC.
La relation de causalité est définie comme l'union des relations $<_{EE}$, $<_{RE}$ et $<_{guard}$. Dans un diagramme de séquence contenant des FC séquentiels (par exemple le DS représenté par la Figure 3.9), la détermination des relations de précédence pour chaque événement du DS n'est pas évidente. En effet, un événement donné peut avoir plusieurs localisations données dans le DS (par exemple à l'extérieur de tous les FC, dans un FC, etc...). Selon son emplacement, ses événements précédents peuvent aussi avoir plusieurs localisations possibles. Pour cette raison, nous avons opté, dans un premier temps, pour procéder cas par cas afin de formaliser les relations de causalités $<_{RE}$ et $<_{EE}$.

Nous pouvons catégoriser les événements qui appartiennent à de tel DS comme suit :
• événements qui sont à l'extérieur du FC ; ils peuvent être situés soit avant tous les FC, soit après soit entre des FC,
• événements qui appartiennent à des opérandes distincts du FC ALT,
• événements qui appartiennent au même opérande.

Notre objectif est d'ordonner toutes les paires d'événements de telle sorte que chaque paire d'événements respecte des conditions qu'on précisera. Á partir de cette catégorisation, nous pouvons déduire les cas possibles des localisations de chaque paire d'événements à ordonner. La Figure 3.10 illustre ces localisations.

Intuition du calcul de la relation de causalité.
La sémantique causale est définie comme l'union des relations $<_{sync}$, $<_{RE}$, $<_{EE}$ et $<_{guard}$.

$$<_{caus} = <_{sync} \cup <_{RE} \cup <_{EE} \cup <_{guard}$$

Pour un diagramme de séquence donné contenant des FC séquentiels, le calcul de la relation de causalité se fait par étapes : nous appliquons d'abord le relation $<_{sync}$, qui permet de synchroniser l'envoi et la réception pour chaque message. Ensuite, nous omettons tous les FC du diagramme et nous appliquons les relations $<_{RE}$ ou $<_{EE}$. L'omission des FC est justifiée puisque si toutes les gardes des opérandes des FC sont évaluées à $Faux$ aucun opérande n'est exécuté.

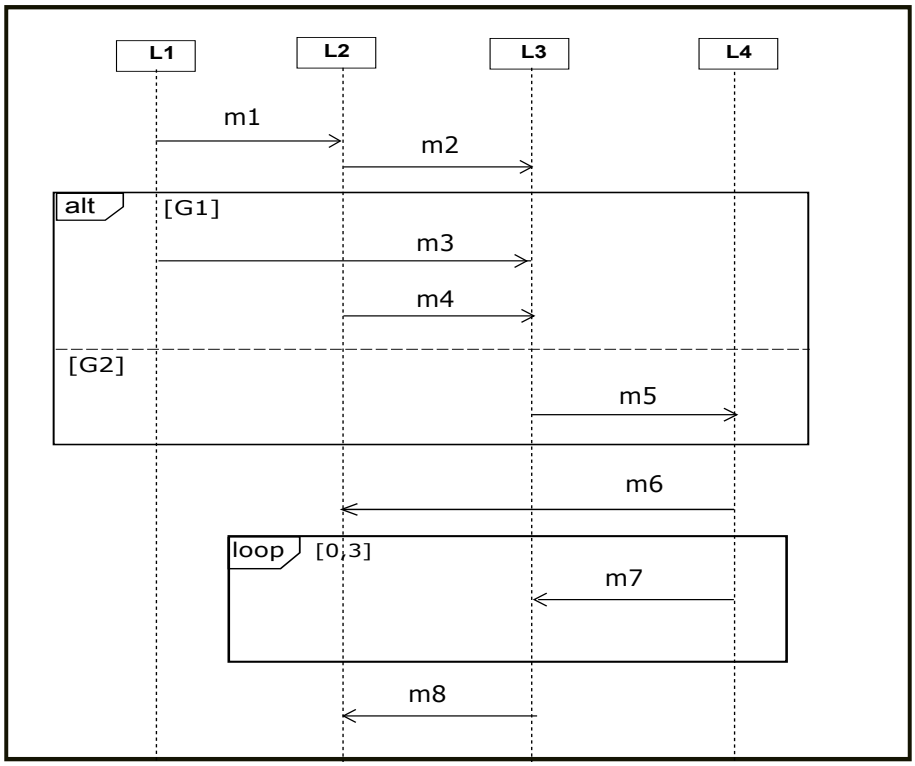

FIGURE 3.9 – Diagramme de séquence avec FC séquentiels

Nous appliquons également les relations $<_{RE}$ ou $<_{EE}$ pour les événements qui appartiennent à un même opérande. Pour le cas du fragment combiné ALT : pour chaque opérande nous omettons les autres opérandes et nous calculons les relations entre les événements de l'opérande considéré et tous les autres événements du diagramme de séquence.

En appliquant cette approche nous évitons le problème de relations aberrantes et de blocage des événements que nous avons déjà mentionné.

3.3.3 Les lois causales

Dans cette section, pour formaliser les lois $<_{RE}$ et $<_{EE}$, nous énonçons les conditions qui doivent être satisfaites par toutes les paires d'événements qui appartiennent à un diagramme de séquence contenant des FC séquentiels.
Nous exploitons les localisations possibles des paires des événements pour dégager les différents cas possibles. Ces cas ne correspondent pas aux localisations possibles. En effet, nous regroupons certaines localisations qui se traitent de façon similaire.
Dans un DS contenant des FC (ALT, OPT, LOOP) qui sont en séquentiel, nous identifions deux types d'événements déclenchables :

[1] $CasI$: les événements qui se situent à l'extérieur de tous les fragments combinés. Pour ces événements, nous identifions deux types possibles de précédents :
- $CasI.1$: le premier cas est quand la précédence est située à l'extérieur du FC (nommés $CasI.1.a$, $CasI.1.b$ dans les formalisations ci-après),
- $CasI.2$: le second cas est quand la précédence est située à l'intérieur d'un opérande quelconque d'un fragment combiné ALT, OPT ou LOOP (nommés $CasI.2.a$, $CasI.2.b$ dans les formalisations ci-après).

[2] $CasII$: les événements qui se situent l'intérieur d'un fragment combiné. Pour ces événements, nous identifions deux types possibles de précédents

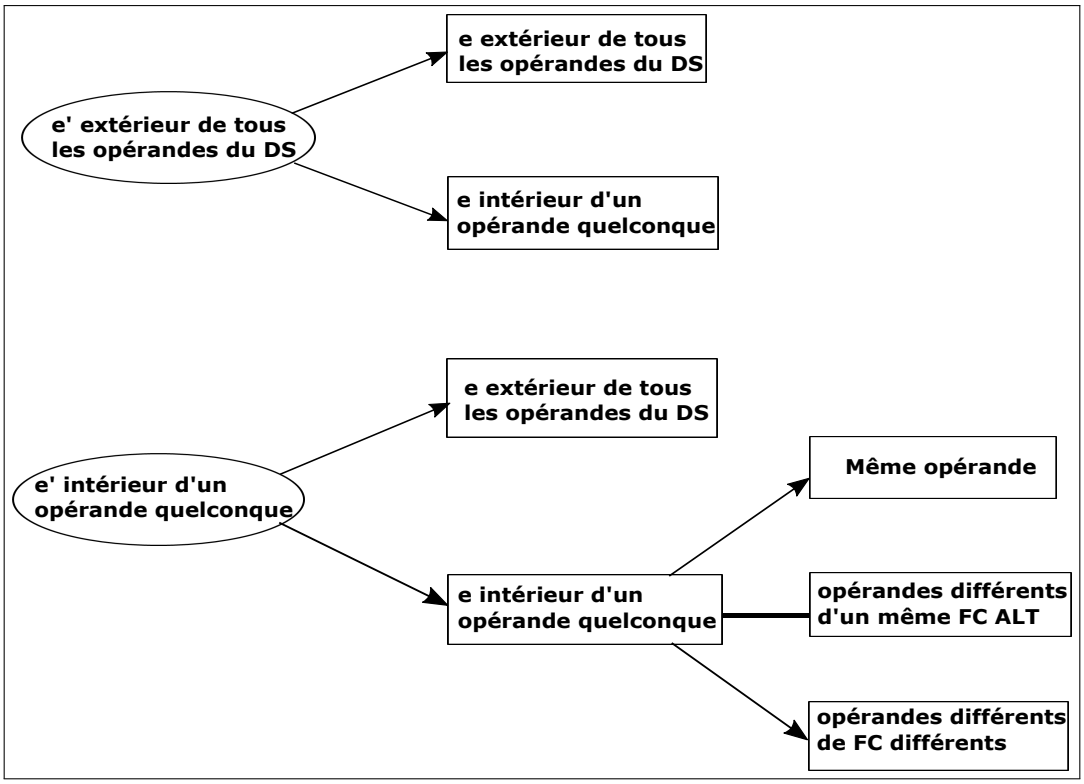

FIGURE 3.10 – Localisations possibles des paires d'événements dans un DS avec FC séquentiels

— $CasII.1$: le premier cas est quand la précédence est située dans le même opérande ou elle est située à l'extérieur de tous les opérandes du diagramme (nommés $CasII.1.a$, $CasII.1.b$ dans les formalisations ci-après),
— $CasII.2$ le deuxième cas est quand la précédence est située à l'intérieur d'un opérande d'un FC différent (nommés $CasII.2.a$, $CasII.2.b$ dans les formalisations ci-après).

Pour chaque cas identifié, nous définissons les conditions ou les lois qui permettent de définir formellement les relations de précédence $< RE$ et $< EE$.

$$<_{RE}= \{(e, e' | CasI.1.a \vee CasI.2.a \vee CasII.1.a \vee CasII.2.a\}$$
$$<_{EE}= \{(e, e' | CasI.1.b \vee CasI.2.b \vee CasII.1.b \vee CasII.2.b\}$$

CasI. **Lois pour les événements déclenchables à l'extérieur de tout FC**

Le $CasI.1$ est quand la précédence se produit à l'extérieur des CF : donc pour chaque paire d'événements à ordonner (e, e'), l'événement e' et son précédent e sont situés à l'extérieur de tous les FC du diagramme de séquence considéré. Le prédicat $CasI.1.a$ pour la relation de précédence $<_{RE}$ est composé de la conjonction de plusieurs clauses qui expriment les conditions sur les événements à ordonner telles que :

— la première condition exprime que les événements e, e' n'appartiennent à aucun FC,
— la deuxième condition exprime que les événements e, e' appartiennent à la même ligne de vie,
— la troisième condition et la quatrième condition expriment respectivement que l'événement e est un événement événement de réception et l'événement e' est événement envoyé,

— la cinquième condition exprime que les événements e, e' sont consécutifs,
— la sixième condition exprime que s'il existe un événement e" entre les événements e et e', e" est nécessairement un événement événement de réception ou il appartient à n'importe quel FC, tel que le FC est situé entre e et e' et il peut être omis (lorsque toutes les gardes de ses opérandes sont fausses).
La dernière clause permet de garder uniquement les réceptions qui sont juste avant l'émission et ignore les événements qui sont à l'intérieur du FC.

CasI.1.*a* : la relation $<_{RE}$

$$
\begin{aligned}
&(\forall e)(\forall e')[(e,e') \in (EVT \setminus OP)^2 \wedge \\
&FCT\_l(e) = FCT\_l(e') = l \wedge \\
&e \in ran(FCT\_r) \ \wedge \ e' \in ran(FCT\_s) \wedge \\
&e <^*_{DS,l} e' \wedge \\
&(\forall e")(e" \in EVT \ \wedge \ (e <^*_{DS,l} e" <^*_{DS,l} e') \\
&\Rightarrow e" \in ran(FCT\_r) \vee e" \in OP)]
\end{aligned}
$$

Le prédicat *CasI*.1.*b* pour la relation de précédence $<_{EE}$ est composé de la conjonction de plusieurs clauses qui expriment les conditions sur les événements à ordonner telles que :

— la première condition exprime que les événements e, e' n'appartiennent à aucun FC,
— la deuxième condition exprime que les événements e, e' appartiennent à la même ligne de vie,
— la troisième condition exprime que les deux événements sont des événements d'émission,
— la quatrième condition exprime que les événements e, e' sont consécutifs,
— la cinquième condition exprime que s'il existe un événement e" entre les événements e et e', e" doit nécessairement appartenir à un FC gardé (pouvant être ignoré si la ou les gardes de son ou de ses opérandes sont fausses).

CasI.1.*b* : la relation $<_{EE}$

$$
\begin{aligned}
&(\forall e)(\forall e')[(e,e') \in (EVT \setminus OP)^2 \wedge \\
&FCT\_l(e) = FCT\_l(e') = l \wedge \\
&(e,e') \in (ran(FCT\_s))^2 \wedge \\
&e <^*_{DS,l} e' \wedge \\
&(\forall e")(e" \in EVT \ \wedge \ (e <^*_{DS,l} e" <^*_{DS,l} e') \\
&\Rightarrow e" \in OP))]
\end{aligned}
$$

CasI.2 le second cas est quand la précédence est située à l'intérieur d'un opérande quelconque d'un fragment combiné ALT, OPT ou LOOP : donc pour chaque paire d'événements à ordonner (e, e'), l'événement e' est situé à l'extérieur de tous les FC et son précédent e est situé dans un opérande quelconque du diagramme de séquence considéré.

Le prédicat *CasI*.2.*a* pour la relation de précédence $<_{RE}$ est composé de la conjonction de plusieurs clauses qui expriment les conditions sur les événements à ordonner telles que :

— la première condition exprime que l'événement e est situé dans une opérande quelconque,
— la deuxième condition exprime que l'événement e' n'appartient à aucun FC,

— la troisième condition exprime que les événements e et e' appartiennent à la même ligne de vie,
— la quatrième condition et la cinquième condition expriment respectivement que l'événement e est un événement événement de réception et l'événement e' est un événement d'émission,
— la sixième condition exprime que les événements e, e' sont consécutifs,
— la septième condition exprime que s'il existe un événement e" entre les événements e et e', e" est nécessairement un événement événement de réception ou c'est un événement qui doit appartenir à n'importe quel opérande gardé, qui est différent de l'opérande (OP_i) auquel appartient l'événement e.

CasI.2.*a* : la relation $<_{RE}$

$$
\begin{array}{l}
(\forall e)(\forall e')[e \in OP_i \ \wedge \ e' \in (EVT \setminus OP) \ \wedge \\
FCT\_l(e) = FCT\_l(e') = l \ \wedge \\
e \in ran(FCT\_r) \ \wedge \ e' \in ran(FCT\_s) \ \wedge \\
e <^*_{DS,l} e' \ \wedge \\
(\forall e")(e" \in EVT \ \wedge \ (e <^*_{DS,l} e" <^*_{DS,l} e') \\
\Rightarrow e" \in ran(FCT\_r) \vee e" \in (OP \setminus OP_i))]
\end{array}
$$

Le prédicat *CasI*.2.*b* pour la relation de précédence $<_{EE}$ est composé de la conjonction de plusieurs clauses qui expriment les conditions sur les événements à ordonner telles que :

— la première condition exprime que l'événement e est situé dans une opérande quelconque,
— la deuxième condition exprime que l'événement e' n'appartient à aucun FC,
— la troisième condition exprime que les événements e et e' appartiennent à la même ligne de vie,
— la quatrième condition exprime que les événements e et e' sont des événements d'émission,
— la cinquième condition exprime que les événements e, e' sont consécutifs,
— la sixième condition exprime que s'il existe un événement e" entre les événements e et e', e" est un événement qui doit appartenir à n'importe quel opérande gardé, qui est différent de l'opérande (OP_i) auquel appartient l'événement e.

CasI.2.*b* : la relation $<_{EE}$

$$
\begin{array}{l}
(\forall e)(\forall e')[e \in OP_i \ \wedge \ e' \in (EVT \setminus OP) \ \wedge \\
FCT\_l(e) = FCT\_l(e') = l \ \wedge \\
(e, e') \in (ran(FCT\_s))^2 \ \wedge \\
e <^*_{DS,l} e' \ \wedge \\
(\forall e")(e" \in EVT \ \wedge \ (e <^*_{DS,l} e" <^*_{DS,l} e') \\
\Rightarrow e" \in (OP \setminus OP_i))]
\end{array}
$$

CasII **: lois pour les événements déclenchables à l'intérieur d'un fragment combiné**

Le **Cas II.1** pour chaque paire d'événements (e, e'), les deux événements à ordonner e et e' peuvent être soit tous les deux situés dans un même opérande OP_i soit l'événement e est situé à l'extérieur et son événement précédent e' est situé à l'intérieur d'un opérande OP_i.
Le prédicat *CasII*.1.*a* pour la relation de précédence $<_{RE}$ est composé de la conjonction de plusieurs clauses qui expriment les conditions sur les événements à ordonner telles que :

— la première condition exprime que l'événement e est soit situé à l'extérieur de tous les FC ($e \in (EVT \setminus (OP))$) soit dans un opérande OP_i,
— la deuxième condition exprime que l'événement e' appartient à l'opérande OP_i,
— la troisième condition exprime que les événements e et e' appartiennent à la même ligne de vie,
— la quatrième condition exprime que l' événement e est un événement événement de réception,
— la cinquième condition exprime que l' événement e est un événement d'émission,
— la sixième condition exprime que les événements e, e' sont consécutifs,
— la septième condition exprime que s'il existe un événement e" entre les événements e et e', e" est nécessairement un événement événement de réception ou c'est un événement qui doit appartenir à n'importe quel opérande gardé, qui est différent de l'opérande (OP_i) auquel appartient l'événement e.

CasII.1.*a* : la relation $<_{RE}$

$$
\begin{aligned}
&(\forall e)(\forall e')[e \in (EVT \setminus (OP \setminus OP_i)) \ \wedge \ e' \in OP_i \ \wedge \\
&FCT\_l(e) = FCT\_l(e') = l \ \wedge \\
&e \in ran(FCT\_r) \ \wedge \ e' \in ran(FCT\_s) \ \wedge \\
&e <^*_{DS,l} e' \ \wedge \\
&(\forall e")(e" \in EVT \ \wedge \ (e <^*_{DS,l} e" <^*_{DS,l} e') \\
&\Rightarrow e" \in ran(FCT\_r) \vee e" \in (OP \setminus OP_i))]
\end{aligned}
$$

Le prédicat *CasII*.1.*b* pour la relation de précédence $<_{EE}$ est composé de la conjonction de plusieurs clauses qui expriment les conditions sur les événements à ordonner telles que :

— la première condition exprime que l'événement e est soit situé à l'extérieur de tous les FC ($e \in (EVT \setminus (OP))$) soit dans un opérande OP_i,
— la deuxième condition exprime que l'événement e' appartient à l'opérande OP_i,
— la troisième condition exprime que les événements e et e' appartiennent à la même ligne de vie,
— la quatrième condition exprime que les événements e et e' sont des événements d'émission,
— la cinquième condition exprime que les événements e, e' sont consécutifs,
— la sixième condition exprime que s'il existe un événement e" entre les événements e et e', e" est un événement qui doit appartenir à n'importe quel opérande gardé, qui est différent de l'opérande (OP_i) auquel appartient l'événement e.

CasII.1.*b* : la relation $<_{EE}$

$$
\begin{aligned}
&(\forall e)(\forall e')[e' \in (EVT \setminus (OP \setminus OP_i)) \ \wedge \ e \in OP_i \ \wedge \\
&FCT\_l(e) = FCT\_l(e') = l \ \wedge \\
&(e, e') \in (ran(FCT\_s))^2 \ \wedge \\
&e <^*_{DS,l} e' \ \wedge \\
&(\forall e")(e" \in EVT \ \wedge \ (e <^*_{DS,l} e" <^*_{DS,l} e) \\
&\Rightarrow e" \in (OP \setminus OP_i))]
\end{aligned}
$$

Le *CasII*.2 est quand la précédence est située à l'intérieur d'un autre opérande OP_j.
CasII.2.*a* et *CasII*.2.*b* : pour chaque paire d'événements (e, e'), les deux événements à ordonner

appartiennent à deux opérandes de deux FC différents OP_i et OP_j.

Le prédicat $CasII.2.a$ pour la relation de précédence $<_{RE}$ est composé de la conjonction de plusieurs clauses qui expriment les conditions sur les événements à ordonner telles que :

- la première condition exprime que l'événement e est situé dans un opérande OP_j,
- la deuxième condition exprime que l'événement e' appartient à l'opérande OP_i,
- la troisième condition exprime que les opérandes des événements e et e' appartiennent à des FC qui sont distincts,
- la quatrième condition exprime que les événements e et e' appartiennent à la même ligne de vie,
- la cinquième condition exprime que l' événement e est un événement événement de réception,
- la sixième condition exprime que l' événement e est un événement d'émission,
- la septième condition exprime que les événements e, e' sont consécutifs,
- la huitième condition exprime que s'il existe un événement e" entre les événements e et e', e" est nécessairement un événement événement de réception ou c'est un événement qui doit appartenir à n'importe quel opérande gardé, qui est différent des opérandes auxquels appartiennent les événements e et e' (OP_j et OP_i).

$CasII.2.a$: la relation $<_{RE}$

$$
\begin{aligned}
&(\forall e)(\forall e')[e \in OP_j \ \wedge \ e' \in OP_i \wedge \\
&\ F_{op}(OP_j) \neq F_{op}(OP_i) \wedge \\
&FCT\_l(e) = FCT\_l(e') = l \wedge \\
&e \in ran(FCT\_r) \ \wedge \ e' \in ran(FCT\_s) \wedge \\
&e <^*_{DS,l} e' \wedge \\
&(\forall e")(e" \in EVT \ \wedge \ (e <^*_{DS,l} e" <^*_{DS,l} e') \\
&\ \Rightarrow e" \in ran(FCT\_r) \vee e" \in (OP \setminus (OP_i \cup OP_j)))]
\end{aligned}
$$

Le prédicat $CasII.2.b$ pour la relation de précédence $<_{EE}$ est composé de la conjonction de plusieurs clauses qui expriment les conditions sur les événements à ordonner telles que :

- la première condition exprime que l'événement e est situé dans un opérande OP_j,
- la deuxième condition exprime que l'événement e' appartient à l'opérande OP_i,
- la troisième condition exprime que les opérandes des événements e et e' appartiennent à des FC qui sont distincts,
- la quatrième condition exprime que les événements e et e' appartiennent à la même ligne de vie,
- la cinquième condition exprime que que les événements e et e' sont des événements d'émission,
- la sixième condition exprime que les événements e, e' sont consécutifs,
- la septième condition exprime que s'il existe un événement e" entre les événements e et e', e" est un événement qui doit appartenir à n'importe quel opérande gardé, qui est différent des opérandes auxquels appartiennent les événements e et e' (OP_j et OP_i).

CasII.2.b : la relation $<_{EE}$

$$
\begin{aligned}
&(\forall e)(\forall e')[e \in OP_j \ \wedge \ e' \in OP_i \wedge \\
&F_{op}(OP_j) \neq F_{op}(OP_i) \wedge \\
&FCT\_l(e) = FCT\_l(e') = l \wedge \\
&(e, e') \in (ran(FCT\_s))^2 \wedge \\
&e <^*_{DS,l} e' \wedge \\
&(\forall e")(e" \in EVT \ \wedge \ (e <^*_{DS,l} e" <^*_{DS,l} e) \\
&\Rightarrow e" \in (OP \setminus (OP_i \cup OP_j)))]
\end{aligned}
$$

3.4 Insuffisances de la première extension pour les diagrammes de séquences avec des fragments combinés imbriqués

L'approche que nous avons proposée pour redéfinir les relations de causalité $<_{RE}$ $<_{EE}$ se base essentiellement sur les différentes localisations que peut avoir deux événements à ordonner dans un DS avec des fragments combinés ALT, OPT et LOOP qui sont disposés séquentiellement. Certes, l'hypothèse que nous avons émise sur la disposition séquentielle des FC nous a facilité l'identification des localisations possibles des paires d'événements. Ces localisations nous ont permis de dégager les différents cas possibles et par conséquent, nous avons donné les formules des relations pour chaque cas permettant d'ordonner toutes les paires des événements dans le DS. Cependant dans les cas pratiques, nous pouvons avoir des comportements complexes des systèmes qui peuvent être modélisés avec des FC imbriqués. Nous avons voulu procéder de la même manière pour déterminer les relations de précédence pour un DS comportant des FC imbriqués. Nous nous sommes confrontés à plusieurs difficultés. Pour éviter les cas complexes, nous avons émis quelques hypothèses assez restrictives. En effet, nous avons considéré les FC (ALT, OPT et LOOP) ainsi que leurs différentes combinaisons possibles, (ALT, ALT), (OPT, OPT), (ALT, LOOP), (ALT, OPT), (OPT, ALT), (LOOP, ALT), (LOOP, LOOP), tout en limitant la profondeur d'imbrication des FC à deux niveaux. Cependant, malgré toutes ces restrictions, nous avons eu une explosion combinatoire de cas possibles, d'autant plus que la détermination des différentes localisations possibles pour les paires des événements dans un DS avec des FC imbriqués s'avère très complexe et non évidente.

Dans la seconde partie de ce chapitre, nous allons montrer comment nous avons surmonté ces difficultés, en proposant une nouvelle formalisation des DS qui est basée sur la structure de données *arbre*, qui a permis de capturer la structure hiérarchique des FC imbriqués ; d'autre part nous avons défini des relations et des fonctions permettant de faciliter la définition de nouvelles lois génériques qui permettent de déterminer les relations de précédence de chaque événement appartenant à un DS comportant des FC imbriqués avec des profondeurs quelconques.

3.5 Seconde extension de la sémantique causale

3.5.1 Mise à jour de la formalisation des diagrammes de séquence

Dans la seconde extension de la sémantique causale, nous considérons en plus des fragments combinés ALT, OPT, et LOOP de nouveaux fragments combinés PAR et STRICT, qui permettent de modéliser des comportements importants des systèmes distribués. Ces FC peuvent être disposés de façon séquentielle et/ou imbriqués pour modéliser des comportements plus sophistiqués. Ce travail a fait l'objet d'une publication dans une conférence classée [Fat17]. Nous supposons que les opérandes des fragments combinés ne se chevauchent pas, ainsi les opérandes internes ne peuvent pas couvrir plus de lignes de vie que les opérandes externes (la Figure 3.11 représente

un exemple de DS comportant des FC qui se chevauchent : le fragment combiné ALT couvre plus de lignes de vie que le fragment combiné LOOP). Notre approche supporte plusieurs niveaux de hiérarchisation de fragments combinés. Nous ne faisons aucune restriction ni sur le nombre de fragments à combiner, ni sur la profondeur d'imbrication des opérandes.

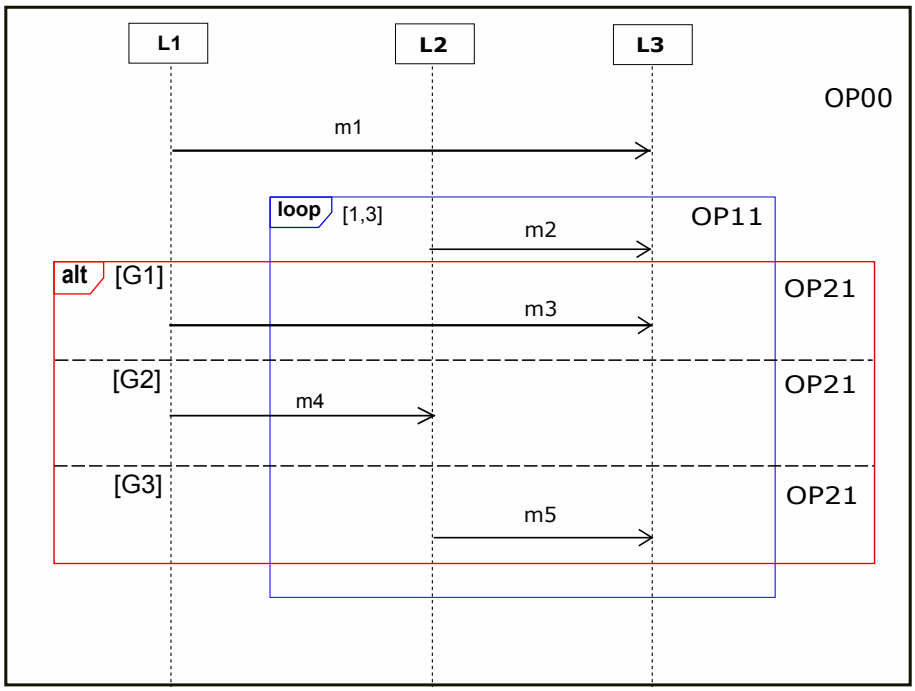

FIGURE 3.11 – Chevauchement de FC

3.5.2 Définition formelle des diagrammes de séquence

Intuitivement, un fragment combiné est vu comme un opérateur et des opérandes. Dans ce qui suit, nous énonçons les nouvelles définitions formelles des diagrammes de séquence. Nous gardons quelques éléments de la formalisation précédente, nous injectons de nouveaux éléments qui permettent de formaliser la structure hiérarchique des fragments combinés imbriqués.

Definition 5. ***(Diagramme de séquence)***
Un diagramme de séquence est un tuple

$$DS : \langle L, M, EVT, FCT\_s, FCT\_r, FCT\_l, OP, F, <_{caus}, tree\_OP \rangle$$

où :
- *L est l'ensemble des lignes de vie,*
- *M est l'ensemble des messages,*
- *EVT est l'ensemble des événements qui est composé des ensembles E_s et E_r qui représentent respectivement l'ensemble des événements d'émission et de réceptions,*
- *FCT_s et FCT_rsont deux fonctions bijectives totales qui permettent d'associer à chaque message respectivement un événement d'émission et un événement de réception,*
- *FCT_l est une fonction surjective totale qui permet d'associer à événement une ligne de vie,*
- *$<_{caus}$ est la relation d'ordre partiel.*
- *OP : est l'ensemble des opérandes du diagramme de séquence considéré,*
- *$F = \{F_1, F_2, ..., F_n\}$ est l'ensemble de n fragments combinés, où $F_i = \langle OP_i, operator_i, L_i \rangle$ est un fragment combiné qui est identifié par ses opérandes, un opérateur et l'ensemble de lignes de vie qu'il couvre.*
- *$tree\_OP$ est une fonction partielle qui permet de structurer le diagramme de séquence sous la forme d'un arbre d'opérandes.*

3.5.3 Fragments combinés imbriqués comme des structures d'arbres

La structure d'arbre, qui est hiérarchique par nature, convient parfaitement pour capturer la structure des diagrammes de séquence contenant des FC imbriqués, permettant de les représenter de façon intuitive. Ainsi, un diagramme de séquence est considéré comme un arbre d'opérandes. Notre formalisation est en concordance avec le méta-modèle du standard UML2.0 à un renommage près de quelques constituants. Nous proposons une extension du méta modèle d'UML qui est conforme à la formalisation que nous proposons qui est basée sur la structure d'arbre, comme illustré dans la Figure 3.12. Nous redéfinissons la classe *Interaction Operand* comme une classe abstraite qui peut être instanciée par : un opérande feuille et ses opérandes imbriqués, telle que cette dernière est reliée à des opérandes frères.

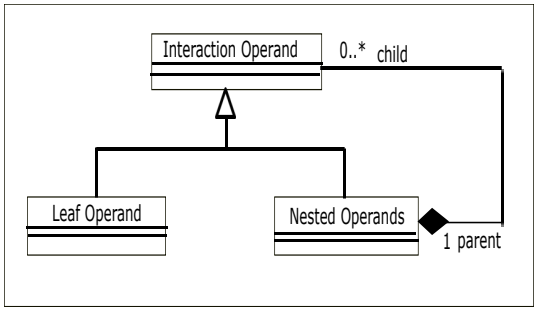

FIGURE 3.12 – Méta-modèle des FC imbriqués

Opérandes des diagrammes de séquence.
Puisque le diagramme de séquence est considéré comme un ensemble d'opérandes reliés entre eux, nous associons à chaque opérande une étiquette (*label*). Deux opérandes qui ont le même parent et le même index i appartiennent au même fragment combiné ; par exemple dans la Figure 3.13, OP_{21}, OP_{22} et OP_{23} ont le même index, donc ils appartiennent au même fragment combiné ALT. Chaque opérande dans le diagramme de séquence possède un *poids*. Par exemple, chaque opérande d'un fragment combiné SEQ, ALT,OPT, PAR et STRICT possède un poids qui est égal à 1. Un opérande d'un fragment combiné LOOP possède un poids qui est égal au nombre maximal d'itération *max* par exemple, dans la Figure 3.13, le poids de l'opérande $OP_{11} = 3$; il doit s'exécuter 3 fois.
Nous supposons que chaque opérande possède un seul premier événement. Nous considérons la totalité du DS comme l'opérande racine que nous notons OP_{00}, nous définissons l'ensemble OP tel que $OP = (\bigcup_{i=\{1..n\}} OP_i) \cup \{OP_{00}\}$ où n est le nombre d'opérandes du DS considéré. La définition générale d'un opérande dans un fragment combiné est donnée ci-après :

Definition 6. ***(Opérande dans un fragment combiné)***
Nous définissons l'ensemble des opérandes dans un fragment combiné F_i par

$$OP_i = \{OP_{i,j=\{1..k\}} \mid OP_{ij} = \langle guard_{ij}, weight_{ij}, EVT\_D_{ij} \rangle\}$$

Où
- *k est le nombre d'opérandes dans le FC Fi,*
- *$guard_{ij}$ est la garde de l'opérande OP_{ij},*
- *$weight_{ij}$ est le poids de l'opérande OP_{ij},*
- *EVT_D_{ij} sont les événements qui sont directement couverts par l'opérande OP_{ij}.*

Pour manipuler les opérandes nous définissons les relations et les fonctions suivantes :
- la relation EVT_D permet de retourner les événements qui sont directement contenus dans chaque opérande.

$$EVT\_D : OP \to \mathbb{P}(EVT)$$

— la relation EVT_G permet de retourner tous les événements contenus dans un opérande y compris ceux qui sont contenus dans ses opérandes fils,

$$EVT\_G : OP \rightarrow \mathbb{P}(EVT)$$

— la fonction *weight* permet de retourner le poids de chaque opérande.

$$weight : OP \rightarrow NAT^{+}$$

— la fonction $first$ permet de retourner le premier événement de chaque opérande. Intuitivement le premier événement d'un opérande est un événement qui n'a pas d'événements qui le précédent dans l'opérande considérée.

$$first : OP \rightarrow EVT$$

$$\begin{aligned} first = &\{(X, e) | X \in OP \ \wedge \ e \in EVT\_G(X) \\ &\wedge \ (\forall \ e')[e' \in EVT \ \wedge \ e' <^{*}_{caus} e \\ &\Rightarrow e' \notin EVT\_G(X)]\} \end{aligned}$$

Nous instancions la définition générale 6 pour les fragments combinés SEQ, ALT, OPT, LOOP, PAR et STRICT ci-après :

Definition 7. ***(Opérande dans un fragment combiné SEQ)***
Un fragment combiné séquentiel contient un seul opérande qui est lié à l'opérateur SEQ.

$$OP_i^{\text{SEQ}} = \{OP_{i1}\}$$

où :

$$OP_{i1} = \langle True, 1, EVT\_D_{i1} \rangle$$

la garde est vraie et le poids vaut 1*.*

Definition 8. ***(Opérande dans un fragment combiné ALT)***
Un fragment combiné ALT F_i *est composé d'un ensemble de k opérandes :*

$$OP_i^{\text{ALT}} = \{OP_{i1}, ..., OP_{ik}\}$$

où :

$$OP_{ij} = \langle (g_{ij} \ \wedge \ A_{ij}), 1, EVT\_D_{ij} \rangle$$

où g_{ij} est une condition qui garde l'opérande OP_{ij}, A_{ij} est un prédicat qui assure que nous avons exclusivement un opérande qui doit être exécuté dans le fragment combiné F_i et le poids de chaque opérande vaut 1*.*

Definition 9. ***(Opérande dans un fragment combiné OPT)***
Le fragment combiné OPT F_i *est composé d'un seul opérande :*

$$OP_i^{\text{OPT}} = \{OP_{i1}\}$$

où :

$$OP_{i1} = \langle g_{ij}, 1, EVT\_D_{ij} \rangle$$

g_{i1} est une condition qui garde l'opérande OP_{ij}.

Definition 10. ***(Opérande dans un fragment combiné LOOP)***
Le fragment combiné LOOP *possède un seul opérande. Ses événements s'exécutent en boucle tant que la garde est évaluée à vrai, en effectuant un nombre minimal de fois (min_{i1}) sans dépasser le nombre maximal indiqué (max_{i1}).*

$$OP_i^{\text{LOOP}} = \{OP_{i1}\}$$

où :

$$OP_{i1} = \langle ((g_{i1} \ \wedge \ B_i) \vee C_i), max_{i1}, min_{i1}, EVT\_D_{i1} \rangle$$

g_{i1} est une contrainte qui garde l'opérande LOOP*, B_i est un prédicat qui indique que l'itération courante est entre la valeur min_i et max_i d'itérations. C_i est un prédicat qui doit être satisfait si l'itération courante est inférieure à la valeur minimale min_i. Le poids correspond à la valeur maximale des itérations.*

Definition 11. ***(Opérandes dans un fragment combiné PAR)***
Un fragment combiné parallèle F_i est composé d'un ensemble de k opérandes :

$$OP_i^{\text{PAR}} = \{OP_{i1}, ..., OP_{ik}\}$$

où

$$OP_{ij} = \langle True, 1, EVT\_D_{ij} \rangle$$

la garde est vraie et le poids vaut 1*.*

Definition 12. ***(Opérandes dans un fragment combiné STRICT)***
Un fragment combiné strict F_i est composé d'un ensemble de k opérandes successifs :

$$OP_i^{\text{STRICT}} = \{OP_{i1}, ..., OP_{ik}\}$$

où

$$OP_{ij} = \langle True, 1, EVT\_D_{ij} \rangle$$

et la garde est vraie et le poids vaut 1*.*

Selon le standard d'UML, la sémantique des interactions des FC est expliquée avec la sémantique d'entrelacement (*interleaving semantics*), qui n'autorise pas l'occurrence simultanée de deux événements déclenchables. De la même manière, nous choisissons une sémantique d'entrelacement pour traiter les comportements alternatifs et concurrents, puisqu'elle est la plus appropriée dans un contexte de système distribué. En effet, si la sémantique permet l'occurrence de deux événements exactement dans le même temps (comme dans la sémantique *true-concurrency* [7]), nous aurons un déclenchement simultané de plusieurs opérandes ayant des gardes vraies or ceci ne permet pas de respecter la sémantique standard du fragment combiné ALT où au plus un seul opérande parmi plusieurs opérandes potentiels doit être choisi.

3.5.4 Encodage d'un diagramme de séquence par une structure d'arbre

Un diagramme de séquence est composé d'un ensemble d'opérandes qui sont liés entre eux, tel que chaque opérande possède au maximum un seul ancêtre direct. La Figure 3.14 représente l'arbre associé au diagramme de séquence de la Figure 3.13. La Figure 3.14 illustre l'arbre associé au DS de la Figure 3.24. Une manière naïve de transformer un DS en un arbre est d'associer un nœud à chaque FC ou opérande. Lors de la construction de l'arbre d'un DS, nous avons toujours un nœud racine qui représente le DS complet. Le parcours de l'arbre est en largeur (*the process is breath-first*). Notons que les opérandes des FC (ALT, PAR, STRICT) sont indépendants, c'est-à-dire qu'ils ont des exécutions disjointes. Par conséquent, pour simplifier la représentation arborescente du DS, nous substituons le nœud qui devrait représenter ces fragments par les nœuds représentant leurs opérandes. Ils sont déplacés vers le niveau supérieur. Cependant, pour les distinguer, les opérandes du même fragment ont leurs index construits avec le même préfixe. Chaque fragment est représenté soit comme un nœud, soit par les nœuds de ses opérandes. Un nœud est associé à chaque FC qui n'a qu'un seul opérande (par exemple LOOP ou OPT). Un FC avec plus d'un opérande (par exemple ALT ou PAR ou STRICT FC) est remplacé par les nœuds associés à ses opérandes.

L'opérande racine OP_{00} n'a pas d'ancêtre. Chaque nœud représente un opérande et admet comme opérandes fils ceux qui lui sont rattachés.
Nous définissons la structure d'arbre associé à un DS comme suit.

7. la sémantique *true-concurrency* est une sémantique sans entrelacement, elle supporte l'occurrence de deux événements en même temps

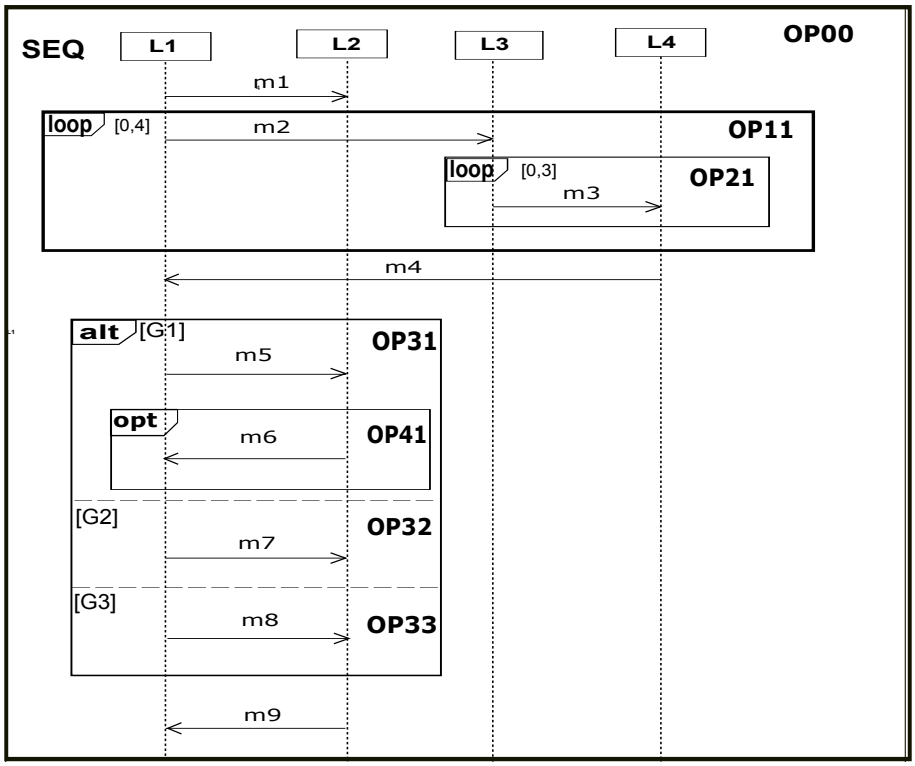

FIGURE 3.13 – Exemple de DS

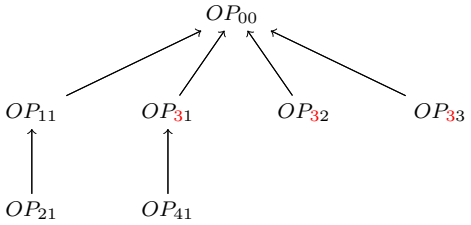

FIGURE 3.14 – Arbre associé au DS de la Figure 3.13

Definition 13. ***(Structure d'arbre pour les opérandes d'un diagramme de séquence)*** *La structure d'arbre (tree_OP) associée à un diagramme de séquence est définie comme une fonction partielle, qui est non cyclique et non réflexive.*

$$tree\_OP : OP \nrightarrow OP$$

La racine est le seul opérande qui n'a pas de parent. Ceci se traduit formellement par :

$$(\forall X)[X \in OP \ \wedge \ X \notin dom(tree\_OP) \ \wedge \ X \in ran(tree\_OP) \Rightarrow X = OP_{00}]$$

Après avoir transformer le DS en un arbre d'opérandes, nous définissons une fonction et des relations qui sont requises pour les formalisations des relations de précédence. Les relations permettent de retourner les localisations de quelques opérandes. Pour associer à chaque opérande toutes les opérandes qui l'englobent (ses opérandes ancêtres dans l'arbre), nous introduisons la relation ancêtre *ancestor*. Pour identifier les opérandes d'un même fragment combiné ALT, PAR, ou STRICT. Nous appelons opérandes *brothers* celles qui appartiennent au même fragment combiné ALT, PAR et STRICT. Dans un arbre donné $tree\_OP = \{OP_{i1}....OP_{ij}\}$, les opérandes *brothers* sont des opérandes qui appartiennent au même niveau et qui ont le même indexe i. Par conséquent, les opérandes d'une même descendance ne sont pas tous nécessairement *brothers*,

puisque certains sont venus du niveau inférieur lorsque nous construisons l'arbre.

• la relation *ancestor* est une relation binaire transitive qui est définie de l'ensemble OP vers l'ensemble OP.

$$ancestor : OP \leftrightarrow OP$$

Pour un opérande X, nous calculons ses ancêtres comme suit :

$$ancestor[\{X\}] = \bigcup_{s\in\{1,..,d\}} \{tree\_OP^s(X)\}$$

où d est la profondeur du nœud OP_{ij} dans l'arbre $tree\_OP$.

• la relation *brother* est une relation transitive binaire qui est définie de l'ensemble des opérandes OP vers l'ensemble des opérandes OP.

$$brother : OP \leftrightarrow OP$$

$$brother = \{(OP_{ij}, OP_{tk}) | (OP_{ij}, OP_{tk}) \in OP^2 \ \wedge \ (i = t \wedge j \neq k)\}$$

Illustration 1 : dans la Figure 3.13, $ancestor[\{OP_{00}\}] = \varnothing$ et $ancestor[\{OP_{41}\}] = \{OP_{31}, OP_{00}\}$.
Illustration 2 : les opérandes $OP_{31}, OP_{32}, OP_{33}$ appartiennent au même fragment combiné ALT, donc ils sont *brothers*. $brother[\{OP_{11}\}] = \varnothing$ et $brother[\{OP_{33}\}] = \{OP_{31}, OP_{33}\}$

Poids d'un événement. Nous surchargeons la fonction *weight* pour associer un poids au chemin entre deux opérandes. Pour deux opérandes X et Y, nous calculons le poids de leurs chemins comme suit :

$$weight : (OP \times OP) \rightarrow NAT^+$$

$$\left\{ \begin{array}{l} weight(X,Y) = 1 \ if \ X = Y \\ weight(X,Y) = \prod\limits_{s\in\{0,..,d\}} weight(tree\_OP^s(Y)) \end{array} \right\}$$

Avec d la longueur du chemin entre l'opérande X et l'opérande Y.
Afin d'associer à chaque événement son nombre maximal d'occurrence, nous surchargeons la fonction *weight*.

$$weight : EVT \rightarrow NAT^+$$

Pour un événement evt, tel que $evt \in EVT\_D(X)$, nous calculons son poids comme suit :

$$\begin{aligned} weight(evt) &= weight(X) * weight(tree\_OP(X)) * \\ &\quad weight(tree\_OP^2(X) * ... * weight(\underbrace{tree\_OP^d(X)}_{OP_{00}})) \\ &= \prod_{s\in\{0,d\}} weight(tree\_OP^s(X)) \ (avec \ d = depth \ of \ X) \\ &= weight(OP_{00}, X) \end{aligned}$$

Illustration : dans la Figure 3.13, les poids associés aux événements associés aux messages $m1, m4, m5, m7, m8$ sont égaux à 1.
Les poids associés aux événements associés au message $m6$ sont égaux à $1 * 1$.
Les poids associés aux événements associés au message $m2$ sont égaux à 4.
Les poids associés aux événements associés au message $m3$ sont égaux à $3 * 4$.

La nouvelle formalisation que nous avons définie dans cette section est utilisée comme base pour la généralisation des relations de précédence de la relation de causalité permettant de calculer l'ordre partiel entre les événements du diagramme de séquence.

Dans cette section, nous avons proposé une structuration des DS avec FC imbriqués sous forme d'un arbre, puis nous avons donner des formalisations qui sont basées sur la structure d'arbre et nous avons défini différentes fonctions et relations qui permettent de naviguer dans cette structure et de déterminer avec évidence les relations requises entre les opérandes. Nous allons exploiter ceci pour généraliser les relations de précédence et de la relation de causalité permettant de calculer l'ordre partiel entre les événements du diagramme de séquence dans la section suivante.

3.6 Généralisation des relations de la sémantique causale étendue

Dans cette section, nous généralisons les règles permettant de calculer, pour chaque événement d'un DS avec des FC imbriqués, ses événements précédents, en exploitant la nouvelle formalisation proposée dans la section précédente sans faire des restrictions syntaxiques sur les FC considérés ; puis nous proposons une approche pour l'évaluation de la garde qui tienne compte de l'indépendance des composants dans le contexte d'un DS modélisant le comportement des systèmes distribués.

3.6.1 Formalisation des relations de précédence $<_{RE}$, $<_{EE}$ et $<_{RR}$

La structuration du DS avec FC imbriqués sous forme d'arbre permet une identification évidente des événements précédents, puisqu'ils sont groupés par opérande, de chaque événement appartenant à ce type de DS. Les relations $<_{Sync}$, $<_{RE}$, $<_{EE}$ et $<_{RR}$ permettent de calculer les relations de précédence pour chaque événement. Dans cette section, nous généralisons ces relations.

La relation de synchronisation ($<_{Sync}$) ordonne l'envoi et la réception de chaque message est inchangeable. Les formalisations des relations $<_{RE}$ et $<_{EE}$ permettent d'ordonner deux événements successifs, qui appartiennent à la même ligne de vie. Nous définissons une nouvelle relation $<_{RR}$ pour considérer certains cas particuliers de l'ordre de réception des événements dans un contexte de composants distribués.

Pour alléger la présentation des formules des relations $<_{RE}$ et $<_{EE}$, nous introduisons trois relations binaires : *not_in_brother*, *succ*1 et *succ*2, dont nous expliquons l'intuition au fur et à mesure.

Il est tout à fait logique que deux événements successifs qui appartiennent à des opérandes distincts d'un fragment combiné ALT ou PAR ou STRICT ne soient pas ordonnés. La relation *not_in_brother* exprime cette intuition : deux événements successifs d'un FC ALT à ordonner ne doivent appartenir ni à des opérandes *brothers* ni à des opérandes tels que dans leurs ancêtres respectifs il existe des opérandes *brothers*.
Deux événements successifs qui appartiennent à des opérandes distincts d'un fragment combiné ALT, PAR, ou STRICT ne devraient pas appartenir au même ancêtre à n'importe quel niveau (ou à des opérandes *brothers*). La relation *not_in_brother* exprime cette intuition.

La relation *not_in_brother* =

$$\begin{aligned}&\textbf{not\_in\_brother}=\{(e,e')|(e,e')\in EVT^2\ \wedge\ (\forall X)(\forall Y)\\&[X\in(ancestor[\{EVT\_D^{-1}(e)\}]\cup\{EVT\_D^{-1}(e)\})\\&\wedge\ Y\in(ancestor[\{EVT\_D^{-1}(e')\}]\cup\{EVT\_D^{-1}(e')\})\\&\Rightarrow(X,Y)\notin brother]\}\end{aligned}$$

Illustration : considérons le DS de la Figure 3.13, les événements $!m7$ et $!m8$ appartiennent respectivement aux opérandes $OP32$ et $OP33$. L'événement $!m7$ ne doit pas être un événement précédent de l'événement $!m8$, puisqu'ils appartiennent respectivement à des opérandes *brothers* : $(OP33\in brother[\{OP32\}])$.
Les événements $?m6$ et $!m7$ appartiennent respectivement aux opérandes $OP41$ et $OP32$.
L'événement $?m6$ ne doit pas être un événement précédent de l'événement $!m7$ puisque l'opérande qui contient l'événement $?m6$ admet un opérande ancêtre qui est *brother* avec l'opérande contenant l'événement $!m7$ $(OP31\in ancestor[\{OP41\}]$ et $OP31\in brother[\{OP32\}])$.

Nous distinguons la succession de deux événements de deux manières, nous exprimons ceci formellement avec deux relations différentes *succ*1 et *succ*2. Ces relations sont utilisées respectivement dans la formalisation des relations $<_{EE}$ et $<_{RE}$.

La relation *succ*1 relie deux événements successifs et qui appartiennent à la même ligne de vie. Cependant, nous acceptons entre eux des événements qui doivent nécessairement appartenir à des opérandes gardés pouvant être omis (les événements successifs n'appartiennent à aucun opérande ancêtre des opérandes des événements considérés).

La relation *succ*1

$$\begin{aligned}&\textbf{succ1}=\{(e,e')|(e,e')\in EVT^2\ \wedge\\&\quad(\exists l)[l\in L\ \wedge\ e<^*_{DS,l}e'\ \wedge\\&\quad(\forall e")[e"\in EVT\ \wedge\ (e<^*_{DS,l}e"\ \wedge\ e"<^*_{DS,l}e')\\&\quad\Rightarrow EVT\_D^{-1}(e")\notin(ancestor[\{EVT\_D^{-1}(e)\}]\\&\quad\cup ancestor[\{EVT\_D^{-1}(e')\}])]]\}\end{aligned}$$

La relation *succ*2 exprime, en plus des conditions et des conséquences qui sont définis dans la relation *succ*1, que nous pouvons accepter entre les événements successifs considérés, des événements qui sont nécessairement reçus.

La relation *succ*2

$$\begin{aligned}&\textbf{succ2}=\ \{(e,e')|(e,e')\in EVT^2\ \wedge\\&\quad(\exists l)[l\in L\ \wedge\ e<^*_{DS,l}e'\ \wedge\\&\quad(\forall e")[e"\in EVT\ \wedge\ (e<^*_{DS,l}e"\ \wedge\ e"<^*_{DS,l}e')\\&\quad\Rightarrow e"\in ran(FCT\_r)\ \vee\\&\quad EVT\_D^{-1}(e")\notin(ancestor[\{EVT\_D^{-1}(e)\}]\\&\quad\cup ancestor[\{EVT\_D^{-1}(e')\}])]]\}\end{aligned}$$

La relation $<_{EE}$ permet d'ordonner deux événements émis, qui satisfont les conditions qui sont exprimées dans les relations *not_in_brother* et *succ*1.

La relation Émission-Émission $<_{EE}$

$$<_{\mathbf{EE}}= \{(e,e')|[(e,e') \in (EVT)^2 \wedge \\ e \in ran(FCT\_s) \ \wedge \ e' \in ran(FCT\_s) \wedge \\ (e,e') \in not\_in\_brother \ \wedge \ (e,e') \in succ1\}]$$

La relation $<_{RE}$ permet d'ordonner deux événements, tels que le premier est un événement reçu et le second est un événement émis ; les événements considérés doivent satisfaire les conditions qui sont exprimées dans les relations *not_in_brother* et *succ*2.

La relation Réception-Émission $<_{RE}$

$$<_{\mathbf{RE}}= \{(e,e')|[(e,e') \in (EVT)^2 \wedge \\ e \in ran(FCT\_r) \ \wedge \ e' \in ran(FCT\_s) \wedge \\ (e,e') \in not\_in\_brother \ \wedge \ (e,e') \in succ2\}]$$

Dans un contexte de système distribué, les composants sont indépendants et la communication entre eux est effectuée selon des protocoles, garantissant chacun d'entre eux des propriétés sémantiques concernant la réception des messages. Dans le cas où le protocole considéré assure un ordre de livraison premier entré premier sorti (FIFO), les réceptions de deux messages provenant de la même ligne de vie sont effectuées dans le même ordre d'émission.
La relation $<_{RR}$ permet de calculer ces relations de précédence.

La relation Réception-Réception $<_{RR}$

$$<_{\mathbf{RR}}= \{(e,e')|[(e,e') \in E_r^2 \wedge \\ (\exists e_1, \exists e_2)[(e_1,e_2) \in E_s^2 \wedge \\ Fct\_s^{-1}(e_1) = Fct\_r^{-1}(e) \wedge \\ Fct\_s^{-1}(e_2) = Fct\_r^{-1}(e') \wedge \\ e_1 <_{EE}^{*} e_2 \ \wedge \ Fct\_l(e_1) = Fct\_l(e_2)]\}$$

Dans ce qui suit, nous allons montrer que la présence des fragments combinés LOOP ainsi que des FC imbriqués comportant des fragments combinés LOOP complique la détermination des relations de précédence pour chaque événement dans ces DS. Nous proposons une approche explicite qui permet de déterminer correctement les relations de précédence dans les DS comportant de telles structures.

3.6.2 Les relations de précédence cachées dans les fragments combinés LOOP

Selon la sémantique standard, l'opérateur FAIBLE SÉQUENCEMENT (WEAK SEQUENCING) entre les itérations d'un fragment combiné LOOP doit être appliqué. Cet opérateur permet un chevauchement entre l'occurrence des événements appartenant à différentes itérations du fragment combiné LOOP. Or ce chevauchement complique la détermination de l'ordre partiel entre les

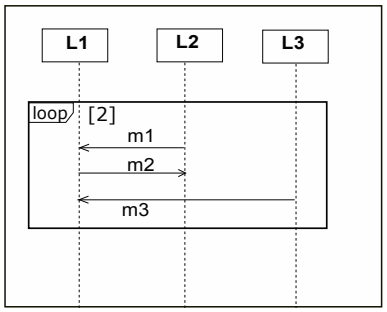

FIGURE 3.15 – Premier exemple du DS avec l'opérande LOOP

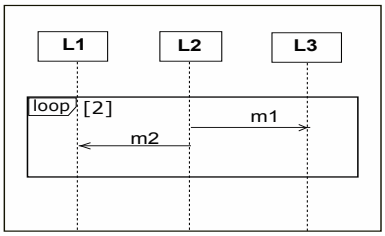

FIGURE 3.16 – Deuxième exemple d'un DS avec une opérande LOOP

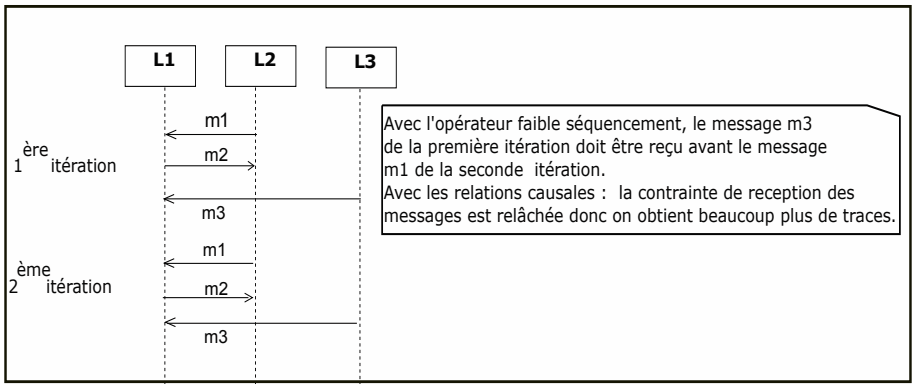

FIGURE 3.17 – Aplatissement de l'opérande LOOP de la Figure 3.15

événements. Ceci explique que la plupart des sémantiques existantes ([Yo06], [Gab08], [Ale04], [Sha11]) optent pour l'interprétation intuitive de l'opérateur LOOP en forçant un strict séquencement entre les itérations, tel que les événements d'une itération courante ne peuvent se produire que si tous les événements de l'itération précédente ont été déclenchés. Cette restriction syntaxique permet d'éviter des cas ambigus et facilite le calcul des relations de précédence entre les événements, cependant elle cause la perte de quelques traces possibles puisque certaines relations de précédence sont supprimées (Figures 3.16 et 3.18).

Dans notre approche, les relations de précédence que nous avons définies permettent aussi un chevauchement entre les événements des différentes itérations. De plus, puisque nous avons relâché la contrainte de l'ordre des événements reçus à partir de lignes de vie indépendantes (à l'exception de quelques cas), nous obtenons beaucoup plus de traces (Figures 3.15 et 3.17).

Les événements qui sont à l'intérieur d'un opérande LOOP peuvent avoir des événements précé-

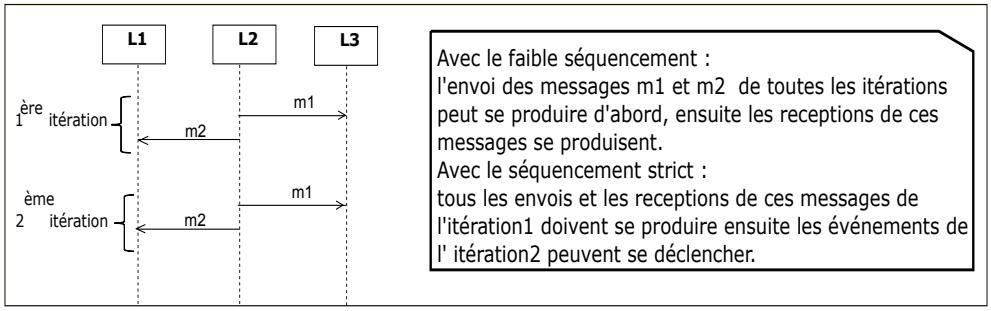

FIGURE 3.18 – Aplatissement de l'opérande LOOP de la Figure 3.16

dents qui sont localisés :

— pour la première itération :
i) soit à l'extérieur de l'opérande LOOP et/ou,
ii) à l'intérieur de l'opérande LOOP de la même itération,
— à partir de la seconde itération : dans les mêmes localisations que pour la première itération et/ou à l'intérieur de l'opérande LOOP des itérations précédentes.

Nous appelons *relations cachées*, les relations entre les événements de l'opérande LOOP de l'itération courante et les événements des itérations précédentes (Figure 3.19). Ces relations apparaissent quand l'opérande LOOP est aplati au moins une seule fois. D'où la nécessité de définir une nouvelle relation $<_{Hcaus}$, dans laquelle nous exprimons les contraintes de précédence entre les événements de l'itération courante et les événements des itérations précédentes.

Pour calculer les relations de précédence cachées, nous proposons les étapes suivantes :
Nous aplatissons l'opérande LOOP une seule fois, quelque soit le nombre des itérations, nous obtenons un diagramme de séquence intermédiaire DS'. Dans DS', nous renommons les opérandes ainsi que les événements de la seconde itération, en ajoutant une simple apostrophe (Figure 3.20). Nous définissons l'ensemble EVT' pour représenter les événements de l'itération suivante. Les relations $<'_{RE}$ et $<'_{EE}$ sont respectivement la relation réception-émission et la relation émission émission qui sont associées au diagramme de séquence DS'. Le renommage est justifié dans l'illustration 2. Nous notons $<_{HcausX}$ les relations de précédence cachées d'un opérande LOOP donnée que nous nommons X ; la formalisation des relations cachées pour un opérande LOOP X est donnée comme suit.

Relation de précédence cachées $<_{HcausX}$

$$<_{\mathbf{HcausX}} = \{(e,e') | e \in EVT \wedge e' \in EVT' \wedge (e,e') \in <'_{RE} \vee (e,e') \in <'_{EE}\}$$

Dans un DS, nous pouvons avoir plusieurs opérandes LOOP qui peuvent être disposés de façon séquentielle ou imbriqués. Dans ce cas, nous appliquons le même traitement en calculant pour chaque opérande LOOP ses relations de précédence cachées.

Illustration 1 : considérons le diagramme de séquence de la Figure 3.19, DS' représente l'aplatissement de l'opérande LOOP une seule fois. Dans DS', dans la première itération, l'événement $!m2$ admet un seul événement précédent $!m1$, qui est localisé à l'extérieur de l'opérande LOOP ; l'événement $!m3$ possède comme événements précédents, l'événement $?m1$ (qui est localisé à l'extérieur de l'opérande LOOP) et $?m2$ (qui appartient à la même itération).

Dans la seconde itération, l'événement $!m2$ a comme événement précédent $!m4$ qui appartient à l'itération précédente, l'événement $!m3'$ a comme précédents les événements $?m4$ et $?m2'$ qui appartiennent respectivement à la première et à la seconde itération de l'opérande LOOP.

Illustration2 : comme nous l'avons mentionné, pour un fragment combiné ALT un seul opérande doit être exécuté, donc il est intuitif que les événements qui appartiennent à des opérandes différents ne doivent pas être ordonnés pour ne pas créer des blocages de quelques événements. Cependant, dans quelques cas particuliers de fragments combinés imbriqués (imbrication de ALT dans un fragment combiné LOOP), nous nous confrontons au problème que des événements appartenant à des opérandes distincts d'un même fragment combiné ALT peuvent avoir des relations de précédence. La Figure 3.21 représente les exécutions possibles du DS (représenté dans la Figure 3.20) contenant des fragments combinés imbriqués ; en effet nous pouvons avoir : le même opérande du fragment combiné ALT qui est exécuté pour toutes les itérations du fragment combiné LOOP (Figures 3.21 (a) et 3.21(b)) ; ou pour chaque itération du fragment combiné LOOP, différents opérandes sont exécutés (Figure 3.21 (c)). Dans ce dernier cas, l'événement

$!m4$ qui appartient au second opérande du fragment combiné ALT possède parmi ses événements précédents, les événements $?m3$ et $?m2$ qui appartiennent à l'itération précédente et qui appartiennent au premier opérande du fragment combiné ALT ce qui est problématique. Ceci justifie le renommage des événements ainsi que des opérandes de l'itération suivante pour contourner ce problème.

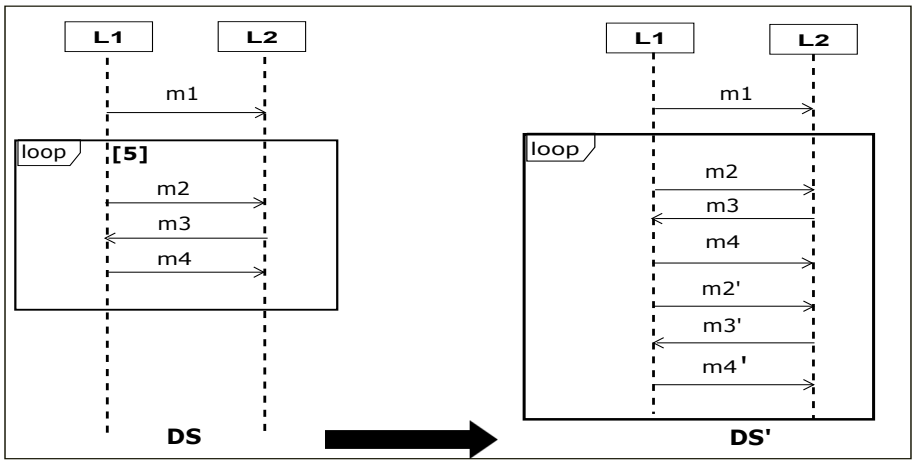

FIGURE 3.19 – Traitement d'un DS avec FC LOOP

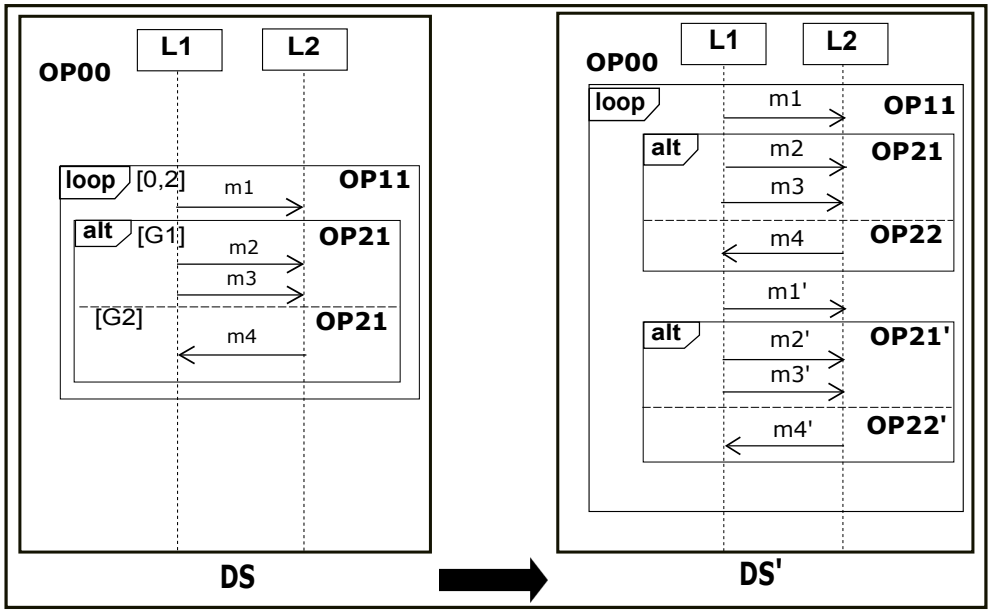

FIGURE 3.20 – Traitement d'un DS avec FC imbriqué

La relation causale est calculée comme suit :

$$<_{caus}=<_{sync} \cup <_{RE} \cup <_{EE} \cup <_{Hcaus}$$

Dans la relation causale $<_{caus}$, nous avons chaque événement d'un DS contenant des FC imbriqués tous ses événements précédents. Pour chaque événement, la vérification de l'occurrence de tous les événements précédents pour un événement conditionne son occurrence. Cependant, son occurrence dépend d'autres contraintes, telle que la valeur de son état, que nous avons besoin de le définir de façon précise et de l'évaluation de la garde s'il s'agit d'un événement qui appartient à un FC gardé.

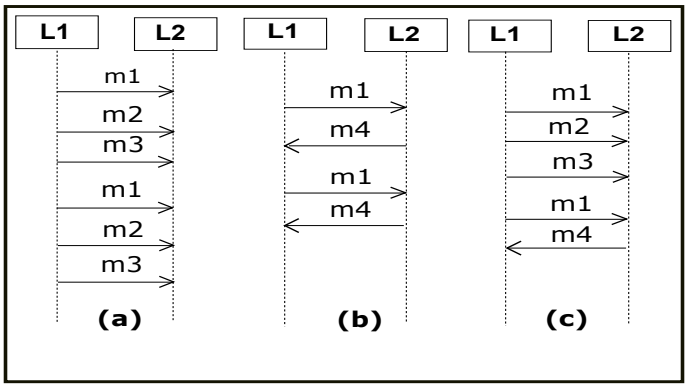

FIGURE 3.21 – Les exécutions possibles du DS de la Figure 3.20

3.7 Évaluation de la garde dans les fragments combinés

Les contraintes d'interactions ou gardes conditionnent l'occurrence des événements qui sont contenus dans le FC. Par conséquent elles ne doivent pas être négligées. Une approche appropriée au système étudié doit être mise en place. En effet, La prise en compte des gardes est un problème complexe (voir Chapitre 2). Il y a deux aspects à considérer pour les gardes : leurs évaluations et l'effet de leurs évaluations sur l'occurrence des événements qui appartiennent au même opérande et des événements des autres opérandes pour des FC bien déterminés (par exemple le FC ALT et notamment des FC imbriqués comportant au moins un FC ALT) : c'est le problème de synchronisation que nous le traitons ultérieurement.

Il y a plusieurs aspects à considérer pour les gardes. Ils concernent la manière ou la méthode de leurs évaluations. Ceci inclut comment et qui doit évaluer les gardes, quand et combien de fois l'évaluation des gardes doivent être faites.

Rappelons que la définition fournie par la sémantique standard est ambiguë. En effet la seule indication est que la garde doit être évaluée avant l'occurrence du premier événement. Ce dernier n'étant pas indiqué clairement. Nous proposons une approche pour l'évaluation de la garde qui est fidèle à la sémantique standard. Nous avons défini de façon précise le premier événement. Nous considérons l'évaluation de la garde comme une opération interne qui est invisible de l'environnement. Par conséquent, nous utilisons des événements fictifs (événements τ) pour traiter le problème de l'évaluation de la garde et le problème de synchronisation. Pour chaque opérande, chaque événement fictif appartient à la même ligne de vie que le premier événement de l'opérande considéré ; il est placé juste avant le premier événement.

Pour chaque opérande gardée, la garde est évaluée une seule fois par la ligne de vie de son événement fictif.

Pour un FC ALT, la garde de chaque opérande est évaluée une fois par l'événement fictif correspondant. Si nous avons plusieurs opérandes avec des gardes vraies simultanément et dont ses événements satisfont les contraintes de précédence, un seul événement fictif va être choisi de façon non-déterministe. Ceci est garanti par notre sémantique qui est une sémantique d'entrelacement (*interleaving semantics*) (i.e deux événements qui sont déclenchables ne peuvent pas se produire en même temps). Dans ce qui suit, nous fournissons la définition formelle des événements fictifs ainsi que les relations de précédence qui sont causées ou créées par leurs introductions dans le DS.

3.7.1 Définition formelle des événements fictifs

Pour chaque opérande gardé OP_{ij}, nous définissons deux événements fictifs positif et négatif (Figure 3.22) puisqu'ils ont vont être utilisés à des fins différents que l'on précisera au fur et à mesure.

L'événement fictif positif (τ_{ij}^{+}) se déclenche lorsque la garde et évaluée à vraie et il permet aux

événements de l'opérande correspondant de se déclencher ; l'événement fictif négatif (τ_{ij}^{-}) se déclenche lorsque la garde et évaluée à faux et il permet d'empêcher l'occurrence des événements de l'opérande concerné. Nous définissons quelques ensembles et fonctions qui sont utilisés par la suite.

• Fic_pos et Fic_neg représentent respectivement les ensembles d'événements fictifs positifs et négatifs tels que $Fic = Fic\_pos \cup Fic\_neg$,

• nous définissons deux fonctions bijectives partielles qui permettent d'associer à chaque opérande respectivement ses événements fictifs positives et négatives.

— $Fct\_\tau\_pos : OP \rightarrowtail\!\!\!\!\!\rightarrow Fic\_pos$,

— $Fct\_\tau\_neg : OP \rightarrowtail\!\!\!\!\!\rightarrow Fic\_neg$.

• $Fct\_l\tau : Fic \rightarrow L$ est une fonction qui permet d'associer à chaque événement fictif sa ligne de vie,

• $<_\tau$: désigne la relation d'ordre partiel qui contient les relations de précédence qui sont causées par l'insertion des événements fictifs. Cette relation est détaillée dans ce qui suit.

Par conséquent, la définition d'un DS est incrémentée avec le tuple suivant :

$$\langle Fic\_pos, Fic\_neg, Fct\_\tau\_pos, Fct\_\tau\_neg, Fct\_l\tau, <_\tau \rangle$$

3.7.2 Les relations de précédence induites par l'introduction des événements fictifs

3.7.2.1 Les relations de précédence pour les FC gardés

Dans un opérande, les événements fictifs sont insérés entre le premier événement et les événements précédents de ce dernier. Ceci engendre de nouvelles relations de précédence entre les événements fictifs et les événements du DS. Ces nouvelles relations de précédence sont calculées à partir d'une relation que nous notons $<_\tau$. Nous décomposons cette relation en trois relations $<_{\tau 1}$, $<_{\tau 2}$, $<_{\tau 3}$ que nous détaillons dans la suite.

3.7.2.1.1 Les relations $<_{\tau 1}$ et $<_{\tau 2}$. Pour un opérande OP_{ij}, les événements fictifs τ_{ij}^{+} et τ_{ij}^{-} sont intercalés entre le premier événement de l'opérande considéré et tous ses événements précédents. Par conséquent, ils héritent tous les événements précédents du premier événement de l'opérande considéré. Ces nouvelles relations de précédence sont calculées à partir de la relation que nous notons $<_{\tau 1}$ qui est formalisée comme suit.

Relation de précédence $<_{\tau 1}$

$$<_{\tau 1} = \{(e, e') | e \in EVT \ \wedge \ e' \in Fic \ \wedge \ (\exists X) \ [(X \in OP) \wedge (e' = Fct\_\tau\_pos(X) \ \vee \ e' = Fct\_\tau\_neg(X)) \ \wedge \ ((e, first(X)) \in <_{caus})]\}$$

De plus, l'événement fictif positif τ_{ij}^{+} devient le seul événement précédent du premier événement, ceci est calculé à partir de la relation $<_{\tau 2}$.

Relation de précédence $<_{\tau 2}$

$$<_{\tau 2} = \{(e, e') | e \in Fic\_pos \ \wedge \ (\exists X) \ [X \in OP \ \wedge \ e = Fct\_\tau\_pos(X) \ \wedge \ (e' = first(X))]\}$$

Illustration 1 : dans la Figure 3.22, selon la relation de précédence $<_{EE}$, l'événement $!m1$ précède l'événement $!m2$ (qui est le premier événement de l'opérande LOOP) ; en insérant les événements fictifs ($\tau+$ et $\tau-$) pour évaluer la garde du FC LOOP, l'événement $!m1$ devient l'événement précédent des deux événements fictifs $\tau+$ et $\tau-$.

Illustration 2 : dans la Figure 3.22, τ^+ devient l'unique événement précédent de l'événement $!m2$.

3.7.2.1.2 La relation $<_{\tau 3}$. Dans un FC imbriqué, nous supposons que l'évaluation de la garde d'un opérande fils (*child*) doit être faite après une évaluation positive de l'opérande parent. Ceci est conforme avec l'hypothèse que nous avons émise, dans laquelle nous avons supposé que chaque opérande possède un seul premier événement.
En outre, si la garde de l'opérande parent est évaluée négativement, ses événements y compris ceux de ses opérandes fils sont ignorés. Cependant, comme illustré dans la Figure 3.23, le cas où plusieurs opérandes imbriqués ont un seul premier événement est un cas possible. Comme nous l'avons mentionné, chaque événement fictif hérite les événements précédents du premier événement de l'opérande concerné. Mais dans ce cas, les événements précédents de l'événement fictif de l'opérande parent sont les mêmes que ceux de l'événement fictif de son opérande fils. Ceci peut mener à un problème de déclenchement intempestif de l'événement fictif de l'opérande fils avant l'occurrence de l'événement fictif de l'opérande parent. Donc, pour ordonner l'occurrence des événements fictifs dans un FC imbriqué et pour générer les relations de précédence entre eux dans ce cas particulier nous définissons une relation que nous nommons $<_{\tau 3}$, qui est formalisée comme suit.

Relation de précédence $<_{\tau 3}$

$$
\begin{aligned}
<_{\tau 3} = \{(e, e') | e \in Fic\_pos \ \wedge e' \in Fic \wedge \\
(\exists X)(\exists Y)[(X, Y) \in OP^2 \wedge e = Fct\_\tau\_pos(X) \wedge \\
(e' = Fct\_\tau\_pos(Y) \vee e' = Fct\_\tau\_neg(Y)) \wedge \\
first(X) = first(Y) \ \wedge tree\_OP(Y) = X]\}
\end{aligned}
$$

Illustration 3 : dans la Figure 3.23, l'opérande parent (OP_{11}) et son opérande fils (OP_{21}) ont le même premier événement ($!m2$) ; l'événement précédent de l'événement fictif positif (τ_{11}^+) de l'opérande OP_{11} est le même que celui de l'événement précédent de l'événement fictif positif de l'opérande. Par conséquent l'événement fictif positif de l'opérande fils (τ_{21}^+) peut se déclencher avant l'événement fictif positif (τ_{11}^+) de l'opérande parent. La relation $<_{\tau 3}$ permet d'ordonner le déclenchement des événements fictifs des opérandes imbriqués dans ce cas particulier. $<_{\tau 3} = \{(\tau_{11}^+, \tau_{21}^+)(\tau_{11}^-, \tau_{21}^-)\}$

3.7.2.2 Relations de précédence pour le fragment combiné STRICT

Pour chaque fragment combiné STRICT qui est composé de N opérandes successifs $OP_{i1}....OP_{iN}$, nous définissons une relation $<_{\tau 4}$ qui permet de calculer les relations de précédences entre chaque événement fictif d'un opérande et tous les événements de l'opérande précédent. La relation $<_{\tau 4}$ est formalisée comme suit.

Relation de précédence $<_{\tau 4}$

$$
\begin{aligned}
<_{\tau 4} = \{(e, e') | e \in EVT \ \wedge e' \in Fic \wedge \\
(\forall j)[e \in EVT\_G(OP_{i-1,j}) \wedge \\
e' = Fct\_\tau(OP_{i,j})
\end{aligned}
$$

Pour récapituler, les relations $<_{\tau 2}$, $<_{\tau 3}$, $<_{\tau 4}$ permettent de générer des relations de précédence qui doivent être ajoutées à la relation de causalités $<_{caus}$, cependant les relations $<_{RE}, <_{EE}, <_{Hcaus}$ doivent être surchargées avec la relation $<_{\tau 1}$. De là, nous définissons la relation $<_{causG}$ comme suit.

$$<_{causG}=<_{Sync} \cup <_{REG} \cup <_{EEG} \cup <_{HcausG} \cup <_{\tau 2} \cup <_{\tau 3} \cup <_{\tau 4}$$

Les relations $<_{REG}$, $<_{EEG}$, $<_{HcausG}$ sont obtenues en surchargeant les relations $<_{RE}$, $<_{EE}$, $<_{Hcaus}$ avec la relation $<_{\tau 1}$.

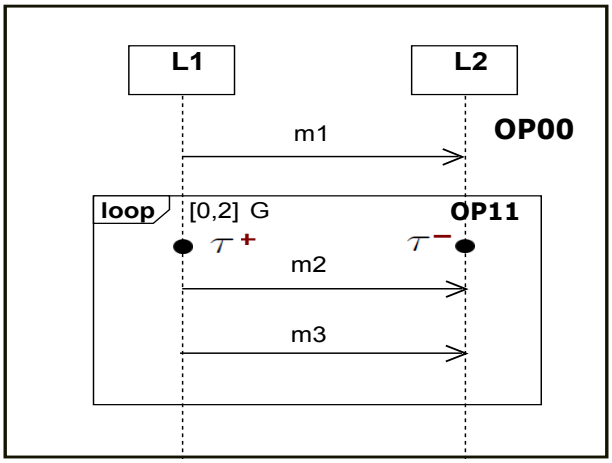

FIGURE 3.22 – Événements fictifs

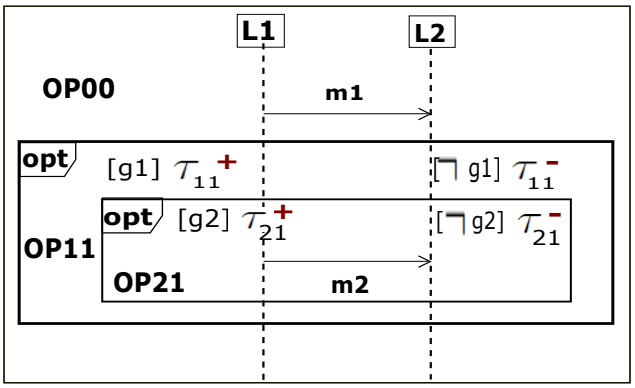

FIGURE 3.23 – DS avec FC imbriqués ayant un même premier événement

3.8 Récapitulatif du calcul des relations de précédence pour un diagramme de séquence d'UML2.X

Pour récapituler le calcul des relations de précédence pour un diagramme de séquence d'UML2.X, nous proposons l'Algorithme 1 ci-dessous.

Compute $R()$ est une fonction qui permet de calculer l'ensemble des couples des événements qui sont ordonnés par la relation correspondante.
$(R =<_{Hcaus}, or <_{Sync}, or <_{EE}, or <_{RE}, or\_\tau 1, or\_\tau 2, or\_\tau 3, or\_\tau 4)$.
La fonction $Override$ $(R1, R2)$ permet de surcharger la relation R1 par la relation R2. La fonction $Unfold$ () permet d'aplatir une seule fois chaque opérande d'un FC LOOP du DS en entrée.

Si nous avons des FC gardés ALT ou LOOP, nous introduisons pour chaque opérande de ces FC des événements fictifs et nous calculons les nouvelles relations de précédence et la relation

Algorithm 1 Calcul de la relation causale : $<_{caus}$.

Require: SD

Ensure: $<_{Sync}, <_{EE}, <_{RE}, <_{Hcaus}, <_{caus}$

1: $<_{Sync} \leftarrow$ COMPUTE SYNC(SD)
2: $<_{EE} \leftarrow$ COMPUTE EE(SD)
3: $<_{RE} \leftarrow$ COMPUTE RE(SD)
4: $<_{Hcaus} \leftarrow \varnothing$
5: **for** $X_i \in OP^{loop}$ **do**
6: $\quad SD' \leftarrow$ UNFOLD(SD, X_i)
7: $\quad <^{i}_{Hcaus} \leftarrow$ COMPUTE HCAUS(SD', X_i)
8: $\quad <_{Hcaus} \leftarrow <_{Hcaus} \cup <^{i}_{Hcaus}$
9: **end for**
10: $<_{caus} \leftarrow <_{Sync} \cup <_{EE} \cup <_{RE} \cup <_{Hcaus}$

de causalité globale en suivant les étapes décrites par l'Algorithme 2. Nous surchargeons la formalisation du DS comme suit :

$$\langle Fic\_pos, Fic\_neg, Fct\_\tau\_pos, Fct\_\tau\_neg, Fct\_l\tau, <_{\tau} \rangle$$

Algorithm 2 Calcul de la relation de causalité globale $<_{causG}$

Require: SD

Ensure: $<_{\tau 1}, <_{\tau 2}, <_{\tau 3}, <_{\tau 4}, <_{causG}$

1: $<_{\tau 1} \leftarrow$ COMPUTE $\_\tau 1(SD,\ Fic\_pos,\ Fic\_neg)$
2: $<_{\tau 2} \leftarrow$ COMPUTE $\_\tau 2(SD,\ Fic\_pos)$
3: $<_{\tau 3} \leftarrow$ COMPUTE $\_\tau 3$(SD)
4: $<_{\tau 4} \leftarrow$ COMPUTE $\_\tau 4$(SD)
5: $<_{caus} \leftarrow$ OVERRIDE $(<_{caus}, <_{\tau 1})$
6: $<_{causG} \leftarrow <_{caus} \cup <_{\tau 2} \cup <_{\tau 3} \cup <_{\tau 4}$

3.9 Expérimentations avec la sémantique causale

Pour illustrer le calcul des relations de précédence que nous avons formellement définies dans ce chapitre, nous considérons un DS modélisant les interactions au sein d'un système de gestion de base de données comme un exemple de DS modélisant les interactions au sein d'un système distribué. Le DS qui est représenté par la Figure 3.24, est constitué des lignes de vie *Client*, *Server* et *Database* qui représentent les composants du système étudié et qui sont indépendantes. Le DS comporte trois FC : deux fragments combinés LOOP, et un fragment combiné ALT. Les deux fragments combinés LOOP sont séquentiels et le fragment combiné ALT est imbriqué dans le second fragment combiné LOOP. Nous avons 7 messages échangés entre les lignes de vie *connect*, *input_query*, *seek_query*, *result_query*, *transmit_query*, *not_found* et *display_not_found*.
Le client doit se connecter (*connect*) au préalable au serveur pour formuler une requête (*input_query*). Il peut formuler au maximum deux requêtes. Le serveur cherche une requête (*seek_query*) dans la base de données, deux alternatives sont possibles : si la requête est trouvée (*result_query*), le serveur transmet le résultat au client (*transmit_query*), sinon (*not_found*) il affiche le message non trouvé (*display_not_found*). Le serveur peut lancer au maximum deux recherches pour une même requête.
Nous représentons les relations de précédence avec les graphes de dépendances représentés par les Figures 3.25 et 3.26 qui sont relatives respectivement aux DS des Figures 3.24 et 3.28. Les Figures 3.27 et 3.28 représentent respectivement l'aplatissement des opérandes LOOP $OP11$ et

$OP21$, pour obtenir les relations de précédence cachées. Dans les graphes de dépendance (Figure 3.25 et Figure 3.26), nous représentons les relations de synchronisation par la couleur bleue, les relations émission-émission par la couleur rouge, les relations réception-émission par la couleur verte et les relations cachées par la couleur violette.

Le calcul des relations de précédence synchronisation, émission-émission, réception-émission, relations cachées, relations induites par l'introduction des événements fictifs donne :

$$\begin{aligned}
&\textbf{CausSync} = \{(e_connect \mapsto e_connect), (e_input_query \mapsto r_input_query),\\
&\qquad (e_seek_query \mapsto r_seek_query), (e_result_query \mapsto r_result_query),\\
&\qquad (e_transmit_result \mapsto r_transmit_result), (e_not_found \mapsto r_not_found),\\
&\qquad (e_display_not_found \mapsto r_display_not_found)\}\\
&\textbf{CausEE}=\{(e_connect \mapsto e_input_query)\}\\
&\textbf{CausRE}=\{(r_input_query \mapsto e_seek_query), (r_seek_query \mapsto e_result_query),\\
&\qquad (r_seek_query \mapsto e_not_found), (r_result_query \mapsto e_transmit_result),\\
&\qquad (r_not_found \mapsto e_display_not_found), (r_connect \mapsto e_seek_query)\}
\end{aligned}$$

$$\begin{aligned}
&\textbf{HCausOP11}=\{(e_input_query \mapsto e_input_query)\}\\
&\textbf{HCausOP21}=\{(e_display_not_found \mapsto e_seek_query), (e_seek_query \mapsto e_seek_query),\\
&\qquad (e_transmit_result \mapsto e_seek_query)\}\\
&\textbf{HCaus} = HCausOP11 \cup HCausOP21\\
&\textbf{HCaus}=\{(e_input_query \mapsto e_input_query), (e_display_not_found \mapsto e_seek_query),\\
&\qquad (e_transmit_result \mapsto e_seek_query), (e_seek_query \mapsto e_seek_query)\}
\end{aligned}$$

$$\begin{aligned}
&\textbf{CausT1EE}=(e_connect \mapsto T_OP11pos), (e_connect \mapsto T_OP11neg)\\
&\textbf{CausT1RE}=(r_input_query \mapsto T_OP21pos), (r_input_query \mapsto T_OP21neg),\\
&\qquad (r_connect \mapsto T_OP21pos), (r_connect \mapsto T_OP21neg),\\
&\qquad (r_seek_query \mapsto T_OP31pos), (r_seek_query \mapsto T_OP31neg),\\
&\qquad (r_seek_query \mapsto T_OP32pos), (r_seek_query \mapsto T_OP32neg)\\
&\textbf{HCausT1}=(e_input_query \mapsto T_OP11pos), (e_input_query \mapsto T_OP11neg),\\
&\qquad (e_display_not_found \mapsto T_OP21pos), (e_display_not_found \mapsto T_OP21neg),\\
&\qquad (e_transmit_result \mapsto T_OP21pos), (e_transmit_result \mapsto T_OP21neg),\\
&\qquad (e_seek_query \mapsto T_OP21pos), (e_seek_query \mapsto T_OP21neg)
\end{aligned}$$

CausT1=$CausT1EE \cup CausT1RE \cup HCausT1$
CausT1=$\{(e\_connect \mapsto T\_OP11pos), (e\_connect \mapsto T\_OP11neg),$
$(r\_connect \mapsto T\_OP21pos), (r\_input\_query \mapsto T\_OP21pos),$
$(e\_display\_not\_found \mapsto T\_OP21pos), (e\_transmit\_result \mapsto T\_OP21pos),$
$(r\_connect \mapsto T\_OP21neg), (r\_input\_query \mapsto T\_OP21neg),$
$(e\_display\_not\_found \mapsto T\_OP21neg), (e\_transmit\_result \mapsto T\_OP21neg),$
$(r\_seek\_query \mapsto T\_OP31pos), (r\_seek\_query \mapsto T\_OP32pos),$
$(r\_seek\_query \mapsto T\_OP31neg), (r\_seek\_query \mapsto T\_OP32neg), \}$
CausT2=$\{(T\_OP11pos \mapsto e\_input\_query), (T\_OP21pos \mapsto e\_seek\_query),$
$(T\_OP31pos \mapsto e\_result\_query), (T\_OP32pos \mapsto e\_not\_found)\}$
CausT3 = $\varnothing$

Pour calculer les relations de précédence $CausEEG$, $CausREG$, $HCausG$ et $CausG$, nous définissons l'ensemble EVT_First qui est l'ensemble des premiers (*first*) événements des opérandes du DS.
Dans la formalisation, nous avons défini les relations $CausEEG$, $CausREG$ et $HCausG$ comme étant la surcharge de chaque relation $CausEE$, $CausRE$ et $HCaus$ par la relation $\tau 1$ ce qui revient à substituer les éléments du co-domaine de la relation considérée qui sont les événements premiers (*first events*) par les événements fictifs (τ *events*).
$EVT\_FIRST = \{e\_input\_query, e\_seek\_query, e\_result\_query, e\_not\_found\}$

CausEEG=$(CausEE \rhd EVT\_FIRST) \cup CausT1EE$
Nous avons $CausEE \rhd EVT\_FIRST = \varnothing$ **donc** $CausEEG = CausT1EE$
$CausRE \rhd EVT\_FIRST =$
$(r\_result\_query \mapsto e\_transmit\_result), (r\_not\_found \mapsto e\_display\_not\_found)$
donc
CausREG = $(r\_result\_query \mapsto e\_transmit\_result), (r\_not\_found \mapsto e\_display\_not\_found),$
$(r\_input\_query \mapsto T\_OP21pos), (r\_input\_query \mapsto T\_OP21neg),$
$(r\_connect \mapsto T\_OP21pos), (r\_connect \mapsto T\_OP21neg),$
$(r\_seek\_query \mapsto T\_OP31pos), (r\_seek\_query \mapsto T\_OP31neg),$
$(r\_seek\_query \mapsto T\_OP32pos), (r\_seek\_query \mapsto T\_OP32neg)$
$HCaus \rhd EVT\_FIRST = \varnothing$

HCausG=$(HCaus \rhd EVT\_FIRST) \cup HCausT1$
donc
HCausG=$HCausT1 = (e\_input\_query \mapsto T\_OP11pos), (e\_input\_query \mapsto T\_OP11neg),$
$(e\_display\_not\_found \mapsto T\_OP21pos), (e\_display\_not\_found \mapsto T\_OP21neg),$
$(e\_transmit\_result \mapsto T\_OP21pos), (e\_transmit\_result \mapsto T\_OP21neg),$
$(e\_seek\_query \mapsto T\_OP21pos), (e\_seek\_query \mapsto T\_OP21neg)$

Par conséquent nous pouvons déduire la relation de causalité globale $CausG$:

$$CausG = CausSYNC \cup CausEEG \cup CausREG \cup HCausG \cup CausT2 \cup CausT3$$

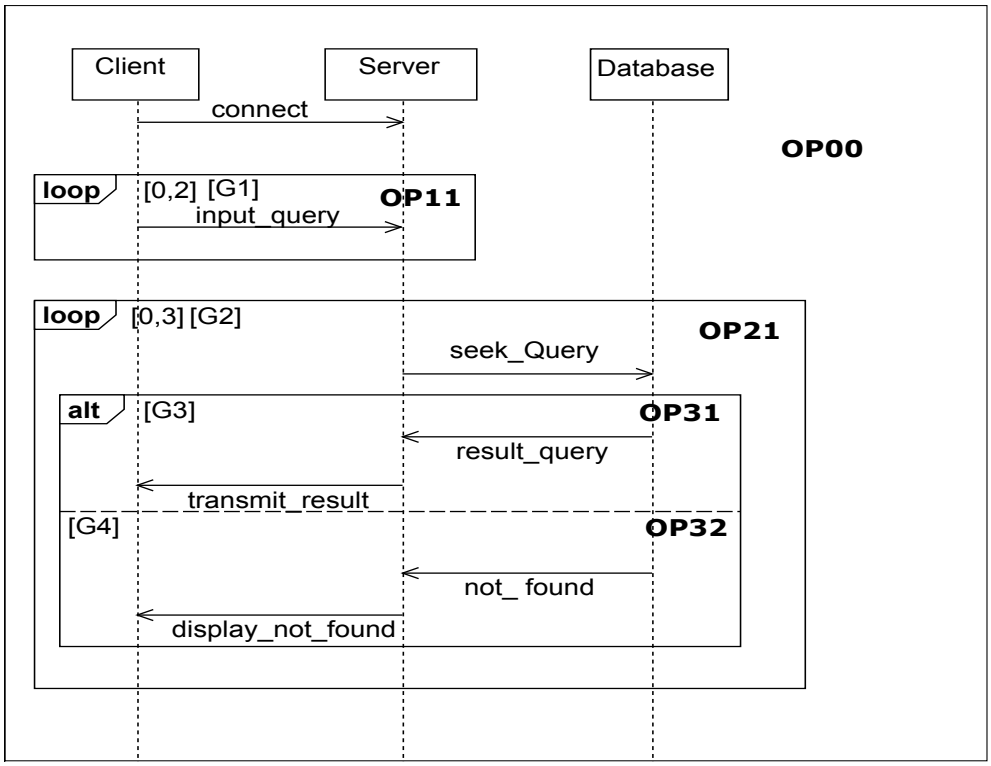

FIGURE 3.24 – DS : recherche d'une requête (*seekquery*)

3.10 Conclusion

Dans ce chapitre, nous avons considéré une sémantique existante qui est destinée aux DS basiques qui modélisent les comportements des systèmes distribués. Nous avons étendu cette sémantique en procédant sur deux étapes :

1. d'abord, nous avons proposer une première extension de la sémantique causale en considérant un nombre limité de fragments combinés en imposant une disposition en séquentiel de ces FC. Ces FC sont ALT, OPT, LOOP, ils modélisent respectivement les comportements alternatifs optionnels et itératifs. Cependant l'utilisation de ces FC en adoptant leurs interprétations standard causent des inconsistances, des problèmes et même des ambiguïtés notamment pour le calcul des relations de précédence entre les événements, problème auquel nous nous intéressons. Ce qui réfutent leur bonne exploitation expliquant les hypothèses restrictives émises dans les approches des travaux existants (voir Chapitre 2 Section 2.3.4) . Même dans notre approche nous avons ignoré, dans la première extension, quelques aspects liés au FC (tels que l'évaluation des gardes, la synchronisation entre les lignes de vie et l'imbrication de FC).
2. puis dans la deuxième extension de la sémantique causale, nous avons considéré de nouveaux FC qui modélisent aussi des comportements importants pour les systèmes distribués. Ce sont les fragments combinés PAR et STRICT qui modélisent des comportement concurrents et stricts. L'idée de l'extension de généraliser la sémantique causale en :

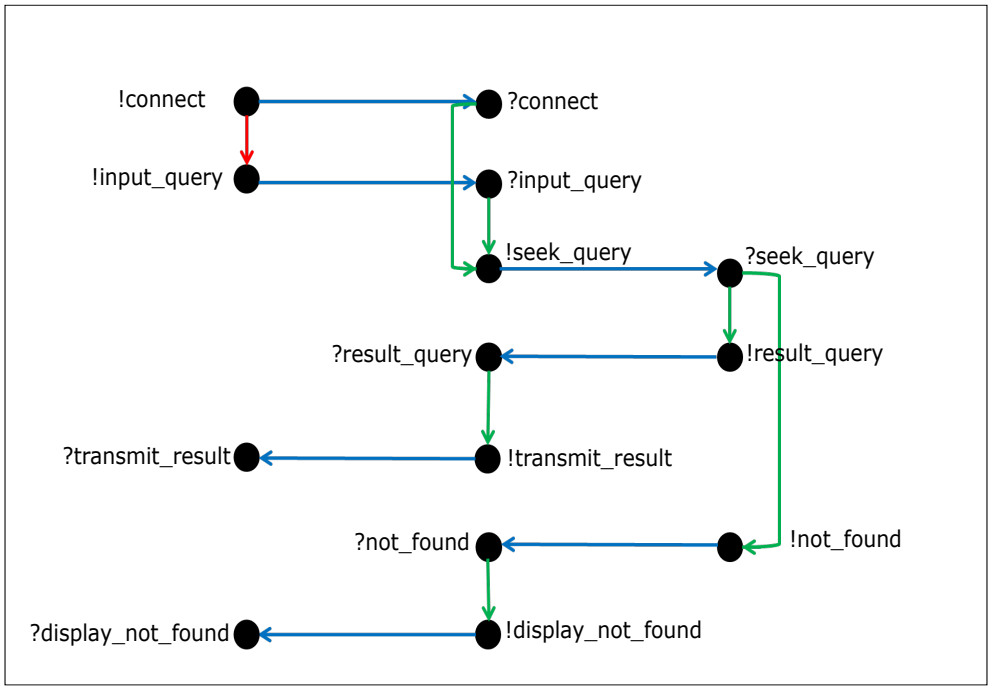

FIGURE 3.25 – Graphe de dépendance associé au DS de la Figure 3.24

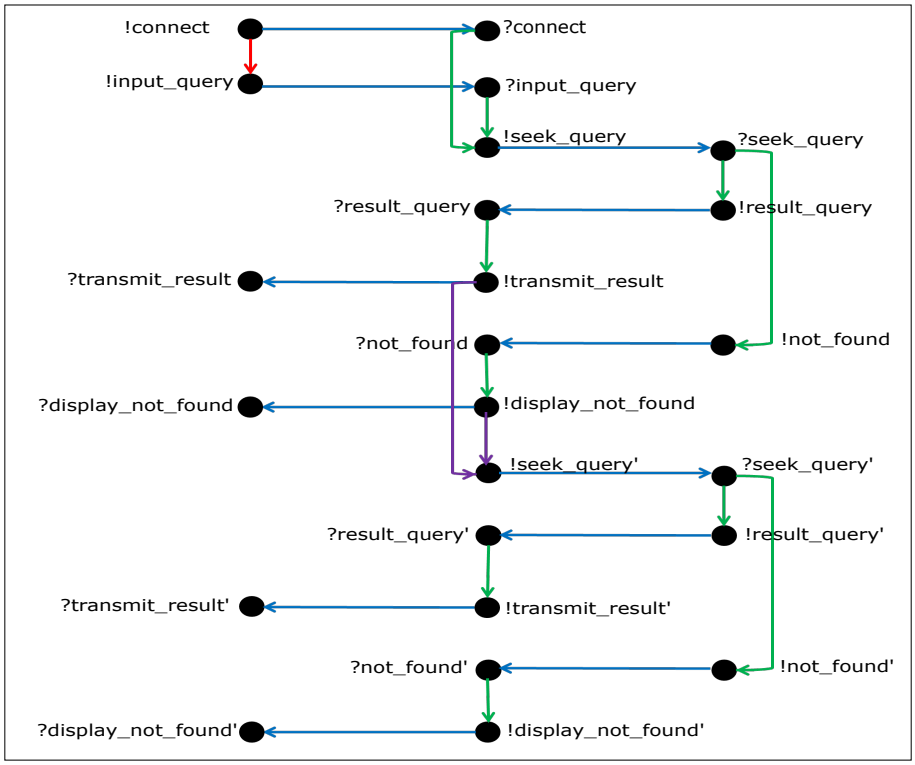

FIGURE 3.26 – Graphe de dépendance associé au DS de la Figure 3.28

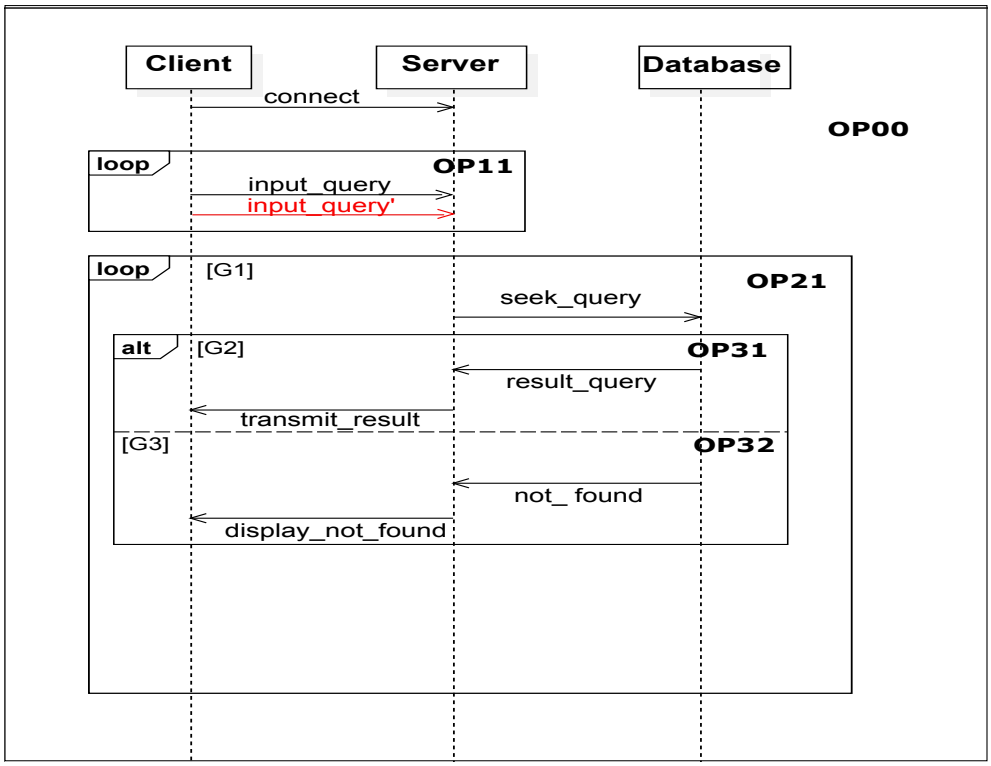

FIGURE 3.27 – Traitement de l'opérande LOOP $OP11$

— proposant des formalisations intuitives des diagrammes de séquence d'UML2.X équipés des fragments combinés (ALT, OPT, LOOP, PAR et STRICT), qui sont basées sur la théorie des ensembles et la structure d'arbre ; nous relâchons l'hypothèse qui impose une disposition en séquentielle des FC ainsi les FC considérés peuvent être disposés n'importe comment imbriqués, séquentiels, imbriqués et séquentiels,
— proposer des relations qui permettent de déterminer les relations d'ordre partiel ou les relations de précédence pour chaque événement appartenant à ce type de DS, en traitant correctement les FC c'est à dire en préservant leurs interprétations standard (i.e sans émettre des restrictions syntaxiques, contrairement aux approches existantes),
— proposer une approche pour l'évaluation de la garde, qui est appropriée aux DS modélisant des interactions entre des composants distribués.

Les différentes formalisations que nous avons définies dans ce chapitre peuvent être utilisées pour des sémantiques modélisant n'importe quel type de système. Selon les exigences du système étudié les relations des précédence peuvent être ajustées. Ceci peut se faire en renforçant ou en affaiblissant certaines contraintes.
Ces travaux ont fait l'objet de publications de trois conférences internationales [Fat15], [Fat17] et [Fat18]

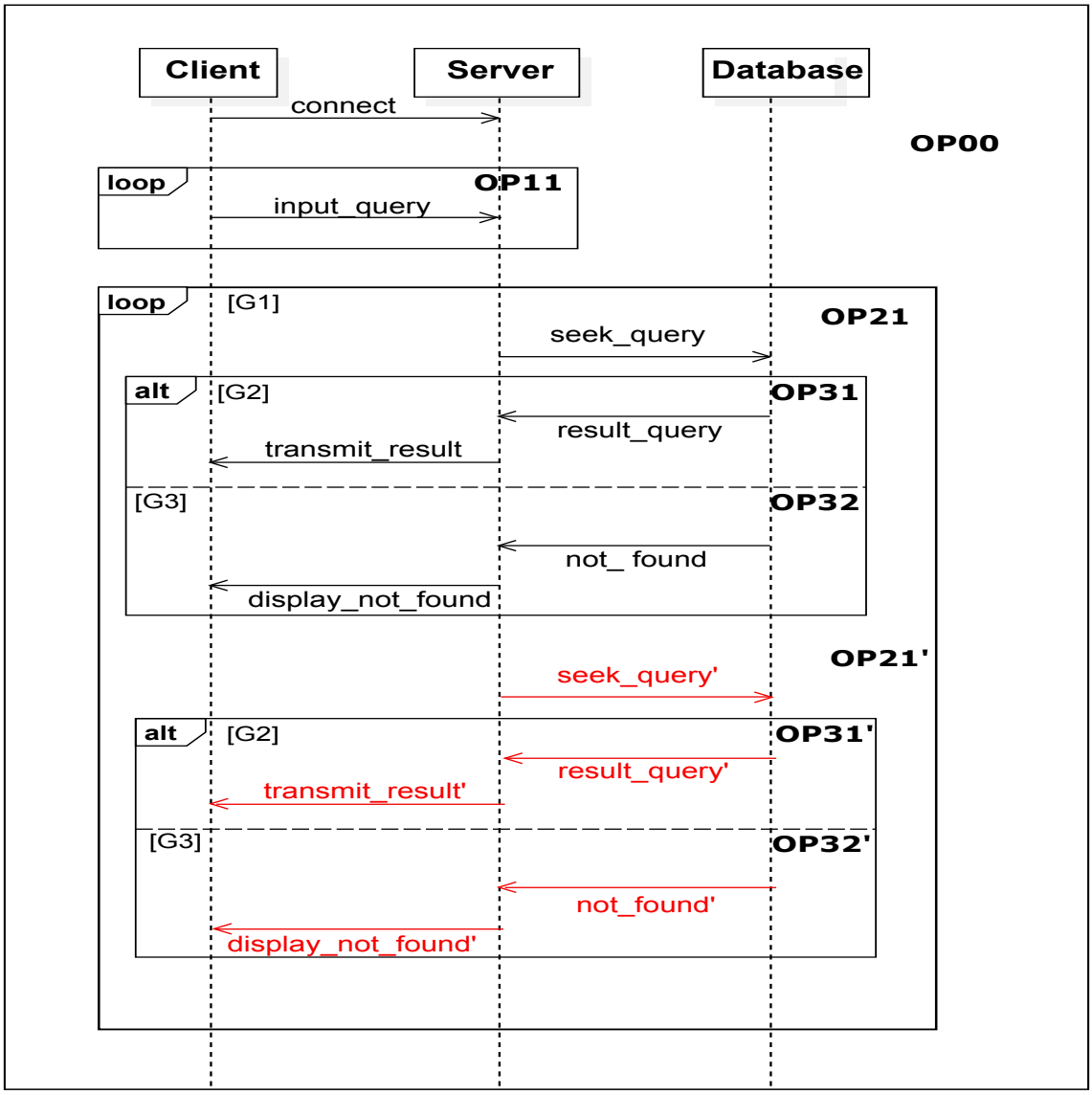

FIGURE 3.28 – Traitement de l'opérande LOOP $OP21$

Chapitre 4

Sémantique opérationnelle

Sommaire

4.1 Introduction

Dans ce chapitre nous nous basons sur les relations de la sémantique causale que nous avons définie dans le chapitre précédent pour définir une sémantique opérationnelle.

L'analyse et la compréhension du comportement d'un diagramme de séquence sont de nature opérationnelle justifient notre choix pour définir une sémantique de nature opérationnelle pour les diagrammes de séquence. Dans la sémantique opérationnelle nous explicitons et nous formalisons les règles qui régissent l'exécution des événements et qui définissent les comportements complexes possibles d'un système modélisé par un DS comportant des FC imbriqués.

4.2 Préliminaires

En théorie des graphes, le plus petit ancêtre commun de deux nœuds d'un arbre est le nœud le plus bas dans l'arbre (i.e le plus profond) ayant ces deux nœuds pour descendants. Le terme en anglais est *Lowest Common Ancestor* (LCA). Les expressions premier ancêtre commun et plus proche ancêtre commun sont aussi utilisées.

Illustration : considérons l'arbre représenté par la Figure 4.1, soient les opérandes X et Y qui sont représentés par les nœuds colorés respectivement en orange et en bleu. Nous représentons les ancêtres de chaque opérande par la même couleur. Les nœuds représentant les ancêtres en commun des deux opérandes sont bicolores. Le LCA est représenté par le nœud Z.

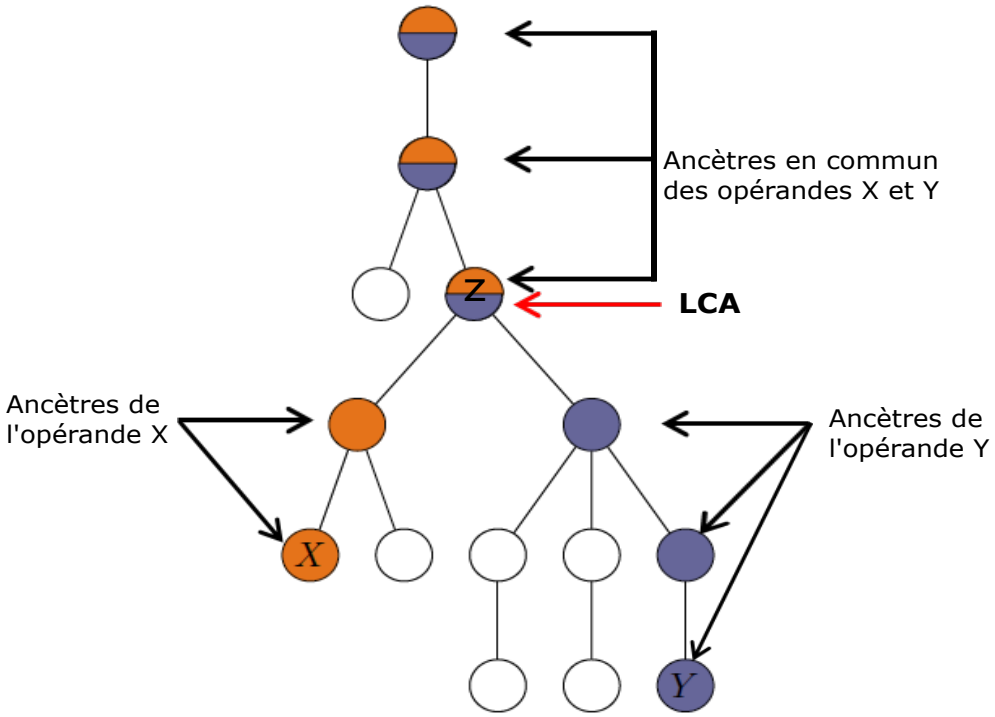

FIGURE 4.1 – Arbre illustrant le LCA des opérandes X et Y

4.3 Comportement d'un diagramme de séquence

Le comportement d'un DS est l'ensemble de ses traces. Une trace est constituée par l'ensemble des occurrences des événements dans une exécution (*run*). La vérification de l'occurrence des événements précédents pour chaque événement conditionne son occurrence. Or ceci est insuffisant, en effet l'occurrence d'un événement peut être sujette à d'autres conditions telle que la valeur de son état, la valeur de la garde de l'opérande auquel il appartient.

4.3.1 L'état d'un événement

Un événement qui appartient à un DS basique peut avoir deux états basiques qui sont évidents : soit il est déclenché soit pas encore. Cependant, ces états basiques ne sont pas suffisants pour exprimer l'état d'un événement qui appartient à un DS ayant des structures imbriqués ; en effet chaque événement qui appartient à de tel DS peut être soit pas encore déclenché (not yet occurred), soit ignoré ou déclenché (occurred) une ou plusieurs fois, ou consommé (consumed). Nous définissons par conséquent la variable *state* comme suit :

$$state : EVT \rightarrow NAT$$

L'état d'un événement est décrémenté à chaque fois où il se déclenche ou s'il est ignoré. Pour décrire l'état d'un événement nous utilisons le vocabulaire suivant :
• not yet occurred quand : $state(e) = weight(e)$, la valeur initiale de l'état chaque événement correspond à son poids qui indique le nombre maximale de son occurrence,
• occurred : si l'événement e est exécuté ou ignoré une ou plusieurs fois quand : $0 < state(e) < weight(e)$,
• consumed : quand $state(e) = 0$.

La notion d'état (*state*) est très importante, en effet à part qu'elle conditionne l'occurrence d'un événement donné (par exemple on décrémente l'état d'un événement à chaque fois où il est

déclenché ou si nous voulons empêcher son occurrence) ; l'état sert aussi à indiquer la localisation d'un événement ; cette information est utile spécifiquement quand nous avons plusieurs FC LOOP imbriqués. Nous utilisons cette notion pour d'autres buts que nous précisons dans ce qui suit.

4.3.2 États d'un diagramme de séquence

Un diagramme de séquence admet plusieurs états possibles durant son exécution. En effet, un DS peut être dans l'un des états suivants :
• un état initial S^0, quand tous ses événements ne sont pas encore déclenchés,
• un état intermédiaire
• un état final quand tous ses événements sont consommés : $state = EVT \times \{0\}$.

Nous exprimons chaque état du DS avec deux variables ($state$, $current\_lifeline$). La première variable exprime l'état de l'événement et la seconde variable exprime la ligne de vie de l'événement en cours.

4.4 Sémantique opérationnelle

Nous définissons une sémantique opérationnelle dans laquelle nous mettons en œuvre des stratégies d'exécutions des événements d'un DS avec des FC imbriqués. Ces stratégies permettent une meilleure compréhension d'un diagramme de séquence et l'analyse de son comportement.
Les stratégies d'exécutions incluent d'une part, l'ordre de l'occurrence des événements du diagramme dans des structures imbriquées ainsi que les conditions sous lesquelles ces occurrences peuvent avoir lieu, d'autre part les effets d'exécution qu'ils produisent.
Notre sémantique opérationnelle, notée $Sem(DS)$, supporte les gardes intuitivement, puisqu'elle est définie comme un *système de transition gardée*. Elle est formellement donnée comme suit :

$$Sem(DS) = \langle S, S^0, \longrightarrow \rangle$$

où S est l'ensemble des états possible du DS, S^0 est l'état initial et $\longrightarrow$ est la relation de transition.

4.4.1 Relation de transition

Pour chaque événement e dans un DS, nous associons la transition suivante :

$$p \xrightarrow{[g]e} q \stackrel{def}{\equiv} ((p,\ [g]\ e,\ q) \in \longrightarrow\ \wedge\ g)$$

Une transition menant le DS d'un état p à un état q correspond à l'occurrence d'un événement du DS. L'événement e peut appartenir à un opérande gardé, dans ce cas son occurrence dépend de la garde (g). Par conséquent, nous associons la variable g à la transition. Un événement est habilité (déclenchable) dans un état p s'il satisfait des conditions de déclenchement qui varient selon le type d'événement considéré.
Dans ce qui suit nous délimitons d'abord les types d'événements qui peuvent exister dans un diagramme de séquence d'UML2.X comportant des FC imbriqués.

4.4.2 Types des événements dans un diagramme de séquence d'UML2.X

Dans un diagramme de séquence, nous avons principalement deux types d'événements :
• ceux qui sont associés aux messages du DS considéré, nous les appelons des événements *normaux*,
• les événements *fictifs* (τ événement). Rappelons que nous avons défini les événements fictifs essentiellement pour l'évaluation des contraintes d'interaction (pour les FC gardés) et pour la synchronisation entre les lignes de vie de différents opérandes. Bien que l'insertion des événements fictifs a engendré de nouvelles relations de précédence que nous avons dû gérer, cependant ils vont nous permettre de définir une forme générique et assez simple des événements du DS. La forme des événements fictifs est relativement complexe et dépendra du FC considéré (ALT ou LOOP ou STRICT). Dans le chapitre précédent, nous avons défini pour chaque opérande gardé d'un FC ALT, LOOP, OPT et STRICT un événement fictif. Grossièrement, selon la valeur de la garde de l'opérande auquel appartient l'événement fictif, il produit des effets d'exécution qui permettent soit de donner la main à d'autres événements de se déclencher soit pour les empêcher de se déclencher. Ceci se fait en agissant sur leurs états respectifs.

Pour alléger les formules des effets d'exécutions que produisent les événements fictifs, nous optons pour définir pour chaque opérande, au lieu d'un seul événement fictif, deux événements fictifs (un positif et un autre négatif) afin de séparer les traitements ; de sorte que l'événement fictif positif se déclenche lorsque la garde est vraie tandis que l'événement fictif négatif se déclenche lorsque la garde est fausse.

Pour chaque opérande gardé, nous définissons deux événements fictifs :
• un événement fictif positif qui permet de déclencher les événements de l'opérande concerné. En plus de cet effet, chaque événement fictif positif de chaque opérande d'un FC ALT a un effet supplémentaire qui consiste à la synchronisation des lignes de vie des autres opérandes du même FC afin d'empêcher l'occurrence des événements des autres opérandes pour prévenir une exécution en parallèle de plusieurs opérandes et l'émergence de comportements non spécifiés dans l'implémentation. Pour se faire l'événement fictif positif (qui se déclenche) d'un opérande d'un FC ALT décrémente l'état de chaque événement qui appartient aux opérandes *brothers*,
• un événement fictif négatif empêche l'occurrence des événements de l'opérande concerné en décrémentant leurs états. Notons qu'un l'événement fictif d'un opérande LOOP doit empêcher l'occurrence des événements de l'opérande pour toutes les itérations si la garde est évaluée à faux dés le début ou pour les itérations restantes (en décrémentant leurs états) si la garde est évaluée à faux durant les itérations.

Pour chaque type et sous type d'événement, nous associons une loi pour le système de transition gardé qui définit sa stratégie d'exécution.

4.4.3 Occurrence des événements

Un événement *evt* se déclenche uniquement lorsque ses *conditions de déclenchement*, que nous les labellisons **CDi**, sont satisfaites conjointement. Une fois déclenché, l'événement produit simultanément des *effets d'exécution*, que nous labellisons **EEi**. Certains **CDi** et **EEi** sont similaires pour tous les événements et d'autres sont spécifiques et varient selon le type d'événement considéré.
Un événement est défini par un ensemble de règles d'inférences de la forme :

$$evt = \frac{CD1 \wedge CD2 \wedge ...CDi}{EE1, EE2, ..., EEi}$$

4.4.3.1 Conditions de déclenchement

Certaines conditions de déclenchement ont une forme simple : ce sont des formules atomiques, quand d'autres conditions sont composées par la conjonction de plusieurs sous conditions.
En effet, certaines conditions doivent être renforcées afin de tenir compte des cas particuliers et pour prévenir certains problèmes résultant de la présence, de la disposition ou de l'imbrication de quelques FC particuliers (par exemple les imbrications comportant des FC LOOP qui engendre des relations de précédences cachées).
Nous explicitons ces problèmes via des illustrations dans ce qui suit.

4.4.3.1.1 Première condition de déclenchement liée à la satisfaction des contraintes de précédence pour les événements qui n'ont pas des relations de précédence cachées. La première condition, que nous labellisons par CD1, nécessaire au déclenchement de chaque événement dans un DS consiste à vérifier que les événements qui le précédent se sont déclenchés.
Rappelons que dans notre sémantique causale, nous procédons dans une étape préliminaire, à transformer le diagramme de séquence considéré sous la forme d'un arbre d'opérandes. Cette transformation nous permet d'identifier les événements précédents de chaque événement qui sont regroupés par opérande. Ensuite les relations de la sémantique causale permettent de déterminer toutes les relations de précédence des événements.
La première condition se base essentiellement sur la comparaison des états relatifs à l'événement considéré et ceux de ses événements précédents. Chaque événement possède un état qui est initialisé à son poids correspondant à son nombre maximal d'occurrence.
Dans un DS contenant des interactions basiques (sans FC) ou dans n'importe quelle imbrication de FC, sans considérer le fragment combiné LOOP, les événements qui appartiennent à ces FC possèdent chacun un poids qui est égale à 1.
Dans une imbrication de FC comprenant au moins un fragment combiné LOOP, le poids étant un terme faisant le produit depuis la racine jusqu'à l'événement. Le poids d'un opérande intermédiaire est un facteur multiplicatif des événements contenus dans les opérandes fils (*childrens*) (illustration 1).
Par conséquent, la comparaison des états de deux événements est basée sur leurs poids relativement à un nœud commun (opérande) ou le premier nœud commun qui englobe ces événements : c'est le premier ancêtre en commun (LCA). En effet, les termes du poids dérivés des ancêtres sont des facteurs multiplicatifs communs.

Illustration 1 : dans la Figure 4.9, considérons les événements $?m1$ et $!m2$ qui appartiennent respectivement aux opérandes $OP21$ et $OP11$, le LCA est l'opérande $OP11$. Par conséquent $weight(?m1) = 3 * 5$ et $weight(!m2) = 5$.
Dans la Figure 4.12, considérons les événements $!m1$ et $!m2$ qui appartiennent respectivement aux opérandes $OP21$ et $OP31$, le LCA est l'opérande $OP11$, donc $weight(!m1) = 5 * 3$ et $weight(!m2) = 5 * 4$.

Par conséquent, pour définir la première de déclenchement, nous raisonnons sur les poids des événements et sur leurs localisations (ils appartiennent à un même opérande ou à des opérandes différentes). Nous procédons cas par cas pour élaborer la formule générale de la première condition CD1. Dans ce qui suit avant d'exposer ces cas, nous donnons l'intuition de la CD1 et les aspects à prendre en compte.

L'événement en cours et ses événements précédents ont un chacun un poids qui est égale à un 1. C'est à dire ces événements peuvent appartenir à un DS basique ou à un même opérande (opérande d'un fragment combiné ALT, OPT, SEQ, STRICT ou PAR) ou encore à une imbrication des FC cités. Pour ces cas il suffit de vérifier dans CD1 de l'événement en cours que les événements précédents ont été consommés (l'état de chacun d'eux est nul).

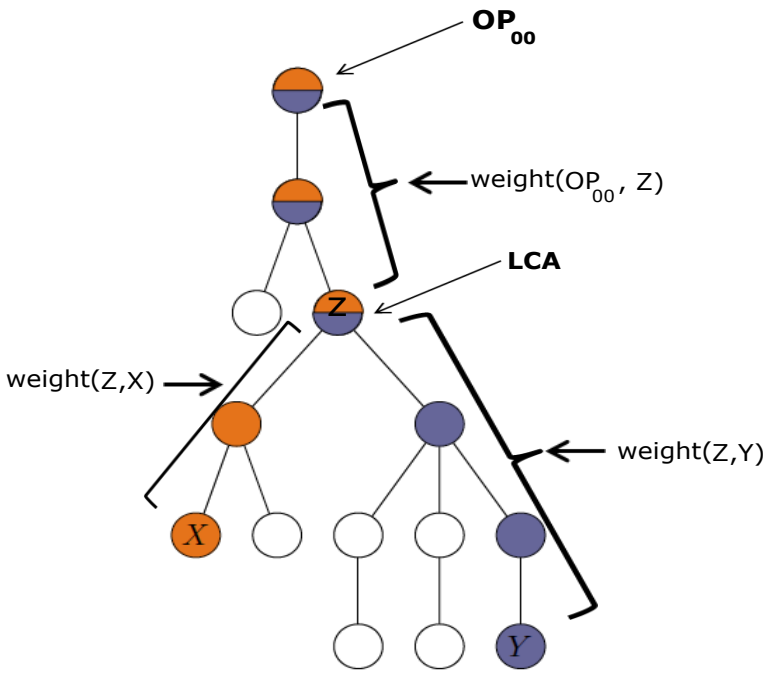

FIGURE 4.2 – Illustration des poids des chemins sur l'arbre

En présence de fragment combiné LOOP, les événements considérés (l'événement en cours et ses précédents) peuvent se déclencher plusieurs fois c'est à dire les événements considérés possèdent des poids qui sont différents de 1.
Plusieurs cas sont envisageables, en effet :

1. les événements considérés (l'événement en cours et ses précédents) ont en plus un même poids :
   — les événements considérés appartiennent soit au même opérande (qui est forcément un opérande LOOP),
   — les événements considérés appartiennent à des opérandes différentes (nous couvrons tous les cas des imbrications possibles des fragments combinés LOOP avec les fragments combinés ALT, OPT, SEQ, STRICT ou PAR) tels que le LOOP est le FC qui englobe l'une de ces FC. Nous pouvons avoir aussi les événements considérés appartiennent à des fragments combinés LOOP ayant le même nombre des itérations mais ces FC sont disposés en séquentiel. Dans ce cas l'événement en cours ne peut se déclencher que si ses événements précédents ont achevé toutes les itérations de l'opérande auquel ils appartiennent.
2. les événements considérés (l'événement en cours et ses précédents) ont des poids différents :
   — l'événement en cours et ses événements précédents appartiennent à des fragments combinés LOOP séquentiels ayant des nombres des itérations différents,
   — l'événement en cours et ses événements précédents appartiennent des fragments combinés LOOP imbriqués. Même dans ces cas nous pouvons avoir de déclenchements différents des événements considérés. En effet nous pouvons avoir le cas où l'événement en cours se déclenche après plusieurs occurrences de ses événements précédents, comme nous pouvons avoir le cas où l'événement en cous se déclenche plusieurs fois pour chaque occurrence de ses événements précédents.

Dans ce qui suit nous illustrons ces cas avec des exemples et nous donnons la formalisation de la condition de déclenchement pour chaque cas afin de donner la CD1 générale.

Pour faciliter les formules, nous supposons que l'événement *evt* possède uniquement un seul événement précédent *e*. Soit l'arbre de la Figure 4.2, où nous considérons deux opérandes X et Y auxquels appartiennent respectivement les événements e et evt : $X = EVT\_D^{-1}(e)$, $Y = EVT\_D^{-1}(evt)$ et Z est le premier ancêtre en commun des opérandes X et Y : $Z = LCA(X, Y)$.

Selon le poids des deux événements e et evt, nous distinguons cinq cas :

[1] Cas1 : le poids de chaque événement e et evt est égale à 1. Dans ce cas, aucun des opérandes X, Y ou Z n'est un fragment combiné LOOP ni leurs ancêtres respectifs ne sont des fragments combinés LOOP.

Nous pouvons avoir deux sous cas sous-jacents qui sont envisageables soit :
i) les deux événements sont localisés dans un même opérande (voir Figure 4.3) :

$$X = EVT\_D^{-1}(e) = EVT\_D^{-1}(evt)$$

ii) ou les deux événements sont localisés dans deux opérandes distincts (voir Figure 4.4) :

$$EVT\_D^{-1}(e) \neq EVT\_D^{-1}(evt)$$

Dans ce cas il suffit de vérifier la condition suivante.

$$state(e) = 0 \ \wedge \ state(evt) = 1 \quad \textbf{(CD11)}$$

Illustration : dans les Figures 4.3 et 4.4, les événements $!m1$ et $!m2$ sont ordonnés selon la relation $<_{EE}$, ils sont respectivement localisés dans le même opérande (Figure 4.3) et dans des opérandes distincts Figure 4.4. Les deux événements possèdent chacun un poids égal à 1. L'événement $!m2$ ne peut se déclencher uniquement si l'événement $!m1$ est consommé. Par conséquent, nous devons vérifier que

$$state(!m1) = 0 \ \wedge \ state(!m2) = 1$$

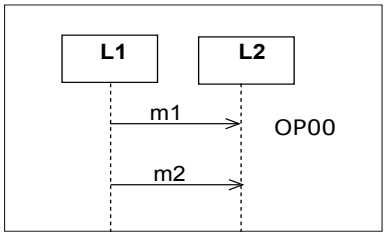

FIGURE 4.3 – DS0 : $!m1 <!m2$

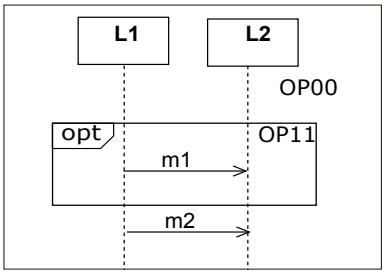

FIGURE 4.4 – DS1 : $!m1 <!m2$

FIGURE 4.5 – Illustration cas1

[2] Cas2 : le poids de chaque événement e et evt est différent de 1. Dans ce cas nous faisons notre étude par rapport au LCA (soit Z) des deux opérandes (X et Y) relatifs aux événements e et evt. En effet, nous distinguons quatre sous cas possibles :

2.1 **Cas2.1** : il n'existe aucun opérande LOOP ni dans le chemin menant de l'opérande Z à l'opérande X, ni dans le chemin menant de Z à Y (i.e. $weight(Z, X) = 1$ et $weight(Z, Y) = 1$),

Deux sous cas sont possibles :
i) les deux événements sont localisés dans un même opérande LOOP (voir Figure 4.6) :

$$X = EVT\_D^{-1}(e) = EVT\_D^{-1}(evt)$$

ii) les deux événements sont localisés dans deux opérandes distincts (Figure 4.7) :

$$EVT\_D^{-1}(e) \neq EVT\_D^{-1}(evt)$$

Nous considérons uniquement les cas où les chemins entre l'opérande Z et l'opérande X et entre l'opérande Z et Y ne contiennent pas un opérande LOOP, (i.e. $weight(Z, X) = 1$ et $weight(Z, Y) = 1$). Dans ce cas, soit l'opérande Z ou au moins un de ses ancêtres est un fragment combiné LOOP.

Illustration1 : dans la Figure 4.6, le poids de chaque événement $!m1$ et $?m1$ est égal à 4, donc chacun d'eux peut se déclencher 4 fois. Pour chaque itération, le message $m1$ peut être reçu uniquement s'il a été envoyé. Ceci conditionne l'occurrence de l'événement $?m1$ et peut être exprimé sous la forme de la condition suivante :

$$state(!m1) < state(?m1)$$

Illustration2 : dans la Figure 4.7, le poids de chacun des événements $!m1$ et $!m2$ est égal à 4, donc chacun d'eux peut se déclencher 4 fois. Pour chaque itération, l'événement $!m2$ ne peut se déclencher que si l'événement $!m1$ s'est déclenché ou s'il a été ignoré.
Ceci conditionne l'occurrence de l'événement $!m2$ et peut être exprimé sous la forme de la condition suivante :

$$state(!m1) < state(!m2)$$

Par conséquent dans ces cas la condition de déclenchement est exprimée comme suit.

$$state(e) < state(evt) \quad \textbf{(CD12)}$$

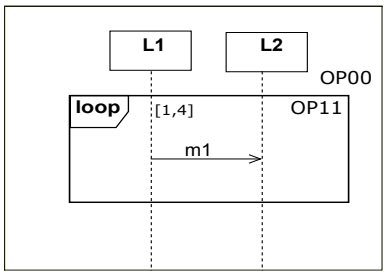

FIGURE 4.6 – DS2 : $!m1 <?m1$

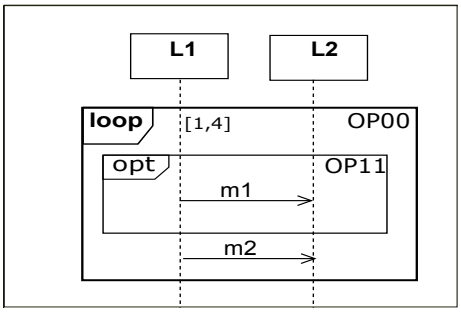

FIGURE 4.7 – DS3 : $!m1 <!m2$

FIGURE 4.8 – Illustration cas2.1

2.2 **Cas2.2** : il existe un opérande LOOP uniquement dans le chemin menant de l'opérande Z à l'opérande X(i.e. $weight(Z, X) > 1$ et $weight(Z, Y) = 1$),

Illustration : soit la Figure 4.9, considérons les événements $?m1$ et $!m2$ qui sont liés causalement par la relation $<_{RE}$; initialement nous avons $state(?m1) = 15$ et $state(!m2) = 5$.

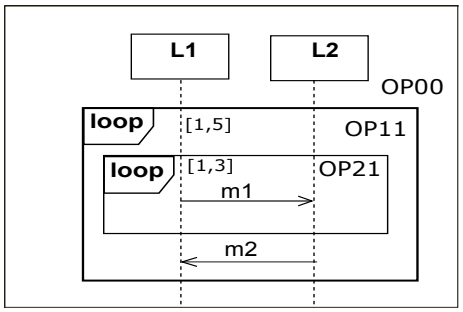

FIGURE 4.9 – DS4 : $?m1 <!m2$

Initialement, $state(?m1) = 15, state(!m2) = 5$	
itération 1 OP11	itération 1 de OP21 : $state(?m1) = 14$ itération 2 de OP21 : $state(?m1) = 13$ itération 3 de OP21 : $state(?m1) = 12$ $\Longrightarrow state(!m2) = 4$
itération 2 de OP11	itération 4 de OP21 : $state(?m1) = 11$ itération 5 de OP21 : $state(?m1) = 10$ itération 6 de OP21 : $state(?m1) = 9$ $\Longrightarrow state(!m2) = 3$
itération 3 de OP11	itération 8 de OP21 : $state(?m1) = 8$ itération 9 de OP21 : $state(?m1) = 7$ itération 10 de OP21 : $state(?m1) = 6$ $\Longrightarrow state(!m2) = 2$
itération 4 de OP11	itération 10 de OP21 : $state(?m1) = 5$ itération 11 de OP21 : $state(?m1) = 4$ itération 12 de OP21 : $state(?m1) = 3$ $\Longrightarrow state(!m2) = 1$
itération 5 de OP11	itération 13 de OP21 : $state(?m1) = 2$ itération 14 de OP21 : $state(?m1) = 1$ itération 15 de OP21 : $state(?m1) = 0$ $\Longrightarrow state(!m2) = 0$

FIGURE 4.10 – Variation des valeurs des états des événements considérés selon l'itération de l'opérande OP11 du DS de la Figure 4.9

Pour chaque itération de l'opérande $OP11$, l'événement $!m2$ peut se déclencher uniquement si l'événement $?m1$ s'est déclenché 3 fois. La Table 4.10 illustre les états possibles des événements $?m1$ et $!m2$.
Par conséquent la condition de déclenchement de l'événement $!m2$ est exprimée comme suit :

$$[state(?m1)/3 < state(!m2)] \ \wedge \ [(state(?m1) mod\ 3 = 0)]$$

En général, pour ces cas la condition de déclenchement de l'événement evt est exprimée sous la forme d'une conjonction de prédicats. Le premier prédicat est une inégalité sur les états de l'événement evt et son événement précédent e tels que l'état de l'événement précédent est pondéré par le coefficient $1/weight(Z, X)$. Le second prédicat permet à l'événement evt d'itérer à condition que son événement précédent e s'est déclenché $weight(Z, X)$ fois.
Nous exprimons cette condition de déclenchement de l'événement evt comme suit.

$$[state(e)/weight(Z, X) < state(evt)] \ \wedge [(state(e) \ \textbf{mod} \ weight(Z, X) = 0)] \quad \textbf{(CD13)}$$

2.3 **Cas2.3** : il existe un opérande LOOP uniquement dans le chemin menant de l'opérande Z à l'opérande Y (i.e. $weight(Z, X) = 1$ et $weight(Z, Y) > 1$),

Illustration : soit la Figure 4.11, considérons les événements $?m1$ et $!m2$ qui sont liés causalement par la relation $<_{RE}$; initialement nous avons $state(?m1) = 5$ et $state(!m2) = 20$. Pour chaque itération de l'opérande $OP11$, l'événement $!m2$ se déclenche 4 fois. La Table 4.1 illustre les états possibles des événements $?m1$ et $!m2$. Par conséquent la condition de déclenchement de l'événement $!m2$ est exprimée comme suit.
$(state(!m2) \ \mathsf{mod}\ 4 = 0) \Longrightarrow (state(?m1) < state(!m2)/4)$

En général, dans ce cas la condition de déclenchement de l'événement *evt* est exprimée comme suit.

$$(state(evt) \textbf{ mod } weight(Z,Y) = 0) \Longrightarrow (state(e) < state(evt)/weight(Z,Y)) \quad \textbf{(CD14)}$$

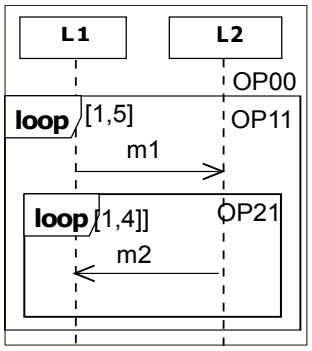

FIGURE 4.11 – DS5 : $?m1 <!m2$

Initialement, $state(?m1) = 5, state(!m2) = 20$	
itération 1 de OP11 : $state(?m1) = 4$	itération 1 de OP21 : $state(!m2) = 19$ itération 2 de OP21 : $state(!m2) = 18$ itération 3 de OP21 : $state(!m2) = 17$ itération 4 de OP21 : $state(!m2) = 16$
itération 2 de OP11 : $state(?m1) = 3$	itération 5 de OP21 : $state(!m2) = 15$ itération 6 de OP21 : $state(!m2) = 14$ itération 7 de OP21 : $state(!m2) = 13$ itération 8 de OP21 : $state(!m2) = 12$
itération 3 de OP11 : $state(?m1) = 2$	itération 9 de OP21 : $state(!m2) = 11$ itération 10 de OP21 : $state(!m2) = 10$ itération 11 de OP21 : $state(!m2) = 9$ itération 12 de OP21 : $state(!m2) = 8$
itération 4 de OP11 : $state(?m1) = 1$	itération 13 de OP21 : $state(!m2) = 7$ itération 14 de OP21 : $state(!m2) = 6$ itération 15 de OP21 : $state(!m2) = 5$ itération 16 de OP21 : $state(!m2) = 4$
itération 5 de OP11 : $state(?m1) = 0$	itération 17 de OP21 : $state(!m2) = 3$ itération 18 de OP21 : $state(!m2) = 2$ itération 19 de OP21 : $state(!m2) = 1$ itération 20 de OP21 : $state(!m2) = 0$

TABLE 4.1 – Variation des valeurs des états des événements considérés selon l'itération de l'opérande OP11 du DS de la Figure 4.11

2.4 **Cas2.4** : il existe un opérande LOOP dans chaque chemin menant de l'opérande Z à respectivement l'opérande X et l'opérande Y (i.e. $weight(Z,X) > 1$ et $weight(Z,Y) > 1$).

Illustration : soit la Figure 4.12, considérons les événements $!m1$ et $!m2$ qui sont liés causalement par la relation $<_{EE}$; initialement nous avons $state(!m1) = 15$ et $state(!m2) = 20$.

Pour chaque itération de l'opérande $OP11$, l'événement $!m1$ se déclenche 3 fois, et l'événement $!m2$ se déclenche 4 fois. Après 3 occurrences de l'événement $!m1$, l'événement $!m2$

se déclenche 4 fois. La Table 4.2 illustre les états possibles des événements $!m1$ et $!m2$. La condition de déclenchement de événement $!m2$ est comme suit.

$$state(!m2) \bmod 4 = 0 \Rightarrow state(!m1)/3 < state(!m2)/4 \ \wedge (state(!m1) \bmod 3 = 0)$$

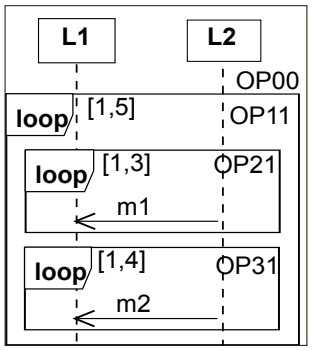

FIGURE 4.12 – DS6 : $!m1 <!m2$

Dans ce cas la condition de déclenchement de l'événement *evt* est exprimée comme suit :

(CD15)

$(state(evt)$ **mod** $weight(Z,Y) = 0) \Rightarrow$
$[state(e)/weight(Z,X) < state(evt)/weight(Z,Y)] \ \wedge \ [(state(e)$ **mod** $weight(Z,X) = 0)]$

Dans les cas que nous avons traités précédemment, nous avons considéré uniquement des événements précédents qui ne sont pas générés par les relations cachées (*hidden relations*) que nous les retrouvons dans de FC contenant des opérandes LOOP ou FC contenant des opérandes LOOP. Par conséquent pour tout événement qui n'admet pas de relations de précédence cachées, la première condition de déclenchement générale $CD1$ n'est autre que $CD15$.

$$CD1 = CD15$$

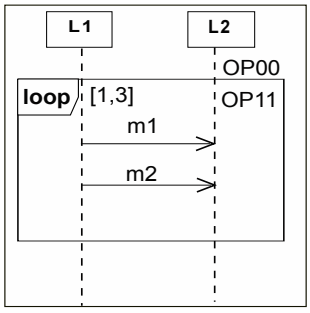

FIGURE 4.13 – DS7 : $!m1 <!m2$

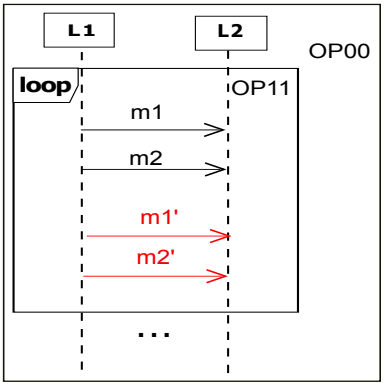

FIGURE 4.14 – DS8 : $!m1 <!m2$

Initialement, $state(!m1) = 15, state(!m2) = 20$	
itération1 de OP11	itération 1 de OP21 : $state(!m1) = 14$ itération 2 de OP21 : $state(!m1) = 13$ itération 3 de OP21 : $state(!m1) = 12$ $\Longrightarrow$ Les événements de OP31 peuvent se déclencher : itération 1 de OP31 : $state(!m2) = 19$ itération 2 de OP31 : $state(!m2) = 18$ itération 3 de OP31 : $state(!m2) = 17$ itération 4 de OP31 : $state(!m2) = 16$
itération 2 de OP11	itération 4 de OP21 : $state(!m1) = 11$ itération 5 de OP21 : $state(!m1) = 10$ itération 6 de OP21 : $state(!m1) = 9$ $\Longrightarrow$Les événements de OP31 peuvent se déclencher : itération 5 de OP31 : $state(!m2) = 15$ itération 6 de OP31 : $state(!m2) = 14$ itération 7 de OP31 : $state(!m2) = 13$ itération 8 de OP31 : $state(!m2) = 12$
itération 3 de OP11	itération 7 de OP21 : $state(!m1) = 8$ itération 8 de OP21 : $state(!m1) = 7$ itération 9 de OP21 : $state(!m1) = 6$ $\Longrightarrow$Les événements de OP31 peuvent se déclencher : itération 9 de OP31 : $state(!m2) = 11$ itération 10 de OP31 : $state(!m2) = 10$ itération 11 de OP31 : $state(!m2) = 9$ itération 12 de OP31 : $state(!m2) = 8$
itération 4 de OP11	itération 10 de OP21 : $state(!m1) = 5$ itération 11 de OP21 : $state(!m1) = 4$ itération 12 de OP21 : $state(!m1) = 3$ $\Longrightarrow$Les événements de OP31 peuvent se déclencher : itération 13 OP31 : $state(!m2) = 7$ itération 14 de OP31 : $state(!m2) = 6$ itération 15 de OP31 : $state(!m2) = 5$ itération 16 de OP31 : $state(!m2) = 4$
itération 5 de OP11	itération 13 de OP21 : $state(!m1) = 2$ itération 14 de OP21 : $state(!m1) = 1$ itération 15 de OP21 : $state(!m1) = 0$ $\Longrightarrow$Les événements de OP31 peuvent se déclencher : itération 17 de OP31 : $state(!m2) = 3$ itération 18 de OP31 : $state(!m2) = 2$ itération 19 de OP31 : $state(!m2) = 1$ itération 20 de OP31 : $state(!m2) = 0$

TABLE 4.2 – Variation des valeurs des états des événements selon les itérations des opérandes du DS de la Figure 4.12

4.4.3.1.2 Première condition de déclenchement liée à la satisfaction des contraintes de précédence pour les événements ayant des relations de précédence cachées.

Dans la Figure 4.13 nous avons la relation de précédence suivante entre les événements $!m1$ et $!m2$ $!m1 <!m2$. À partir de la seconde itération nous avons la relation de précédence cachée $!m2 <!m1$ (voir Figure 4.14) : l'événement $!m2$ devient précédent de l'événement $!m1$.

La Table 4.3 illustre les états possibles des événements $?m1$ et $!m2$. À partir de la seconde

n° itération	$state(!m1)$	$state(!m2)$
1	2	2
2	1	1
3	0	0

TABLE 4.3 – Variation des valeurs des états selon l'itération de l'opérande OP11 du DS de la Figure 4.13

itération, l'événement $!m1$ doit vérifier la condition de déclenchement suivante : $state(!m2) = state(!m1)$

En général : pour chaque événement *evt* d'un opérande LOOP qui admet des relations de précédence cachées dont elles apparaissent dès la deuxième itération de l'opérande LOOP et qui peuvent être localisées soit dans des opérandes *brothers* ou non, nous définissons la condition de déclenchement suivante CD1'.

Lorsque nous avons :
$(\forall e)(\exists X)(\exists Z)[(e, evt) \in <_{Hcaus} \wedge X = EVTD^{-1}(e) \wedge Y = EVTD^{-1}(evt) \wedge Z = LCA(X, Y)$
La condition de déclenchement est exprimée comme suit :

(CD1')

$(state(e)\ \textbf{mod}\ (weight(Z,Y) * weight(Z) <> 0)) \Longrightarrow$
$((state(e)/weight(Z,X) = state(evt)/weight(Z,Y)$

4.4.3.1.3 Autres conditions de déclenchement. La deuxième condition nécessaire au déclenchement d'un événement (CD2) consiste à vérifier qu'il est encore déclenchable : c'est-à-dire il n'a pas encore atteint son nombre maximal d'occurrence (i.e il n'est pas consommé), formellement CD2 est définie comme suit.

$state(evt) \geq 1$ **(CD2)**

4.4.3.1.4 Conditions de déclenchement supplémentaires pour les événements fictifs. Pour les événements fictifs, nous rajoutons une condition de déclenchement qui permet de tester la garde de l'opérande à laquelle appartient l'événement fictif.
• Pour un événement fictif positif, la garde doit être évaluée à vrai :

$G = TRUE$ **(CD3)**

• Pour un événement fictif négatif, la garde doit être évaluée à faux :

$G = FALSE$ **(CD3')**

4.4.3.2 Les effets d'exécutions

Une fois déclenché, un événement produit simultanément plusieurs effets d'exécutions. Nous expliquons séparément les effets d'exécutions des événements normaux et ceux des événements fictifs positifs et négatifs des fragments combinés ALT et LOOP.
Les événements fictifs ont des effets d'exécutions supplémentaires puisqu'ils doivent assurer la synchronisation entre les lignes de vie comme nous l'avons expliqué dans le chapitre précédent.

4.4.3.2.1 Effets d'exécutions des événements normaux.

— Le premier effet d'exécution que produit chaque événement consiste à mettre à jour son état. Ceci s'exprime formellement comme suit.

$$state(evt) := state(evt) - 1 \quad \textbf{(EE1)}$$

— Le deuxième effet d'exécution consiste à mettre à jour la ligne de vie de l'événement courant.

$$current\_lifeline := FCT\_l(evt) \quad \textbf{(EE2)}$$

4.4.3.2.2 Effet d'exécution des événements fictifs : traitement du problème de synchronisation. Les événements fictifs d'un fragment combiné ALT ou d'un fragment combiné LOOP ont des effets d'exécutions additionnels pour traiter le problème de synchronisation entre les lignes de vie. Ce problème est une conséquence de l'évaluation des gardes dans ces FC.

Dans un contexte de composants distribués, une fois que la garde est évaluée, la décision doit être propagée proprement aux autres lignes de vie ; ce qui permet de prévenir l'émergence de comportements non spécifiés qui sont peuvent se produire suite à un déclenchement en parallèle des événements qui appartiennent à des opérandes distincts d'un fragment combiné ALT. Ce raisonnement est cohérent avec la sémantique standard du fragment combiné ALT [Obj15] qui impose l'occurrence des événements qui sont contenus dans un opérande ayant une garde vraie parmi plusieurs opérandes ayant en simultané des gardes vraies. Si aucune garde n'est vraie le FC est omis. Dans un contexte de DS modélisant des comportements d'un système distribué, les lignes de vie sont indépendants et une synchronisation doit être assurée. En outre, pour les fragments combinés LOOP gardés, les événements qui sont contenus dans l'opérande sont exécutés tant que la garde est évaluée à vraie et le nombre maximal des itérations n'est pas atteint. Le fragment combiné LOOP représente une application récursive de l'opérateur faible séquencement (*weak sequencing*), les traces qui contiennent un chevauchement entre l'occurrence des événements des différentes itérations sont possibles (Figure 4.15). Par conséquent, une fois la garde devient fausse, les événements des itérations précédentes qui ne se sont pas déclenchés doivent se déclencher, quant aux événements des itérations restantes, ils ne doivent pas se déclencher.

Illustration : dans la Figure 4.15 nous représentons un DS' qui est sémantiquement équivalent à un DS comportant un fragment combiné LOOP. Dans DS', les itérations du fragment combiné LOOP sont dépliées. L'événement $!m1$ précède l'événement $!m2$ dans toutes les itérations. Le message $m1$ de la seconde itération peut être reçu avant les réceptions des messages $m1$ et $m2$ de l'itération précédente ce qui peut mener au chevauchement de l'occurrence des événements des différentes itérations.

Les événements fictifs vont assurer la synchronisation en agissant sur les états des événements de opérande auquel ils appartiennent ou des autres opérandes. Dans ce qui suit nous donnons la formalisation des effets d'exécutions des événements fictifs.

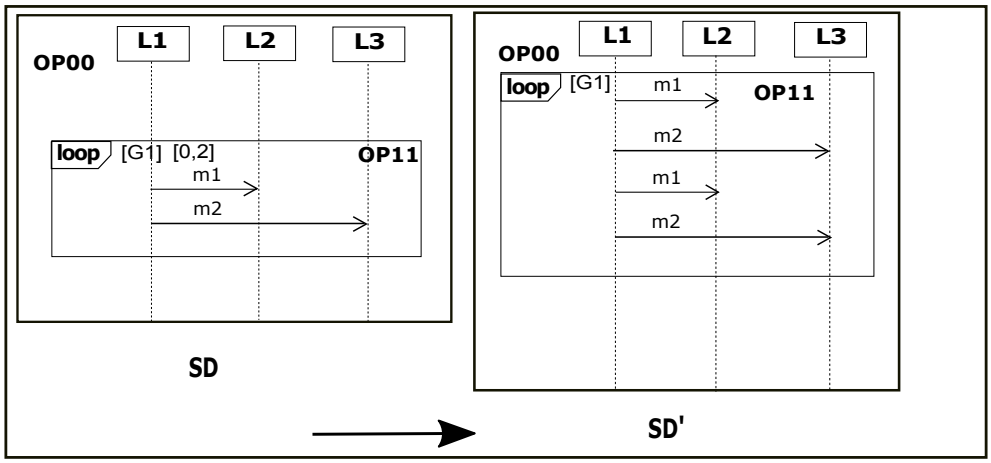

FIGURE 4.15 – Dépliage des itérations du FC LOOP CF

1. Effet d'exécution d'un événement fictif positif d'un fragment combiné ALT
   nous considérons l'événement *evt* comme un événement fictif positif d'un fragment combiné ALT tel que $Y = EVT\_D^{-1}(evt)$. En se déclenchant l'événement *evt* produit les effets d'exécutions suivants qui permettent de :
   • mettre à jour son état de l'événement *evt* en décrémentant son état **EE1** et en mettant à jour sa ligne de vie courante de l'événement courant (**EE2**).
   • mettre à jour l'état de l'événement fictif négatif au même opérande auquel il appartient soit l'opérande Y. Soit evt' l'événement fictif négatif de l'opérande Y tel que : $evt' = Fct\_\tau\_neg(Y)$; formellement cet effet d'exécution s'écrit comme suit.

$$state(evt') := state(evt') - 1 \quad \textbf{(EE1')}$$

   • décrémenter l'état de chaque événement des opérandes *brothers* afin de prévenir leurs déclenchements (c'est-à-dire pour les ignorer). Pour tout événement e appartenant aux opérandes *brothers* de Y, il faut décrémenter son état. L'événement e peut être localisé soit directement dans l'opérande *brother* de l'opérande Y soit dans un des opérandes qui est imbriqué dans l'opérande Y.
   Considérons les opérandes Z et X tels que $Z = brother(Y)$, $X = EVT\_D^{-1}(e)$ et $e \in EVT\_G(Z)$. Pour empêcher l'occurrence de l'événement e, nous décrémentons son état par rapport à son poids relativement à l'opérande Z soit $(weight(Z, X))$.
   Ceci revient formellement à exprimer l'effet d'exécution suivant, **EE3**, dans l'événement *evt* :

$$state(e) := state(e) - weight(Z, X) \quad \textbf{(EE3)}$$

2. Effet d'exécution d'un événement fictif négatif d'un fragment combiné ALT.
   Pour l'événement fictif négatif d'un fragment combiné ALT, nous définissons l'effet d'exécution (**EE3'**) qui permet de décrémenter l'état de tous les événements de l'opérande considéré pour les ignorer.
   Considérons l'événement *evt* comme un événement fictif négatif d'un fragment combiné ALT tel que $Y = EVT\_D^{-1}(evt)$.
   Pour tout événement e appartenant au même opérande Y, il faut décrémenter son état. L'événement e peut être soit localisé directement dans l'opérande Y, ou il peut être localisé dans un opérande qui est imbriqué dans l'opérande Y. Soient $X = EVT\_D^{-1}(e)$

et $e \in EVT\_G(Y)$. Pour empêcher l'occurrence de l'événement e, nous décrémentons son état par rapport à son poids relativement à l'opérande Y ($weight(Y, X)$).
Par conséquent, l'effet d'exécution produit par l'événement fictif négatif *evt* du fragment combiné ALT qui permet d'agir sur les états de tout événement e s'exprime formellement comme suit.

$$state(e) := state(e) - weight(Y, X) \quad \textbf{(EE3')}$$

3. Effet d'exécution d'un événement fictif positif d'un fragment combiné LOOP.
   Considérons l'événement *evt* comme un événement fictif positif d'un fragment combiné LOOP tel que $Y = EVT\_D^{-1}(evt)$. Nous définissons pour l'événement *evt* les effets d'exécution suivants :
   • mettre à jour l'état de l'événement *evt* en décrémentant son état **EE1**,
   • mettre à jour sa ligne de vie courante de l'événement courant (**EE2**),
   • mettre à jour l'état de l'événement fictif négatif du FC Y : $evt' = Fct\_\tau\_neg(Y)$ (**EE1'**)

4. Effet d'exécution d'un événement fictif négatif d'un fragment combiné LOOP
   Pour l'événement fictif négatif d'un fragment combiné LOOP, nous définissons l'effet d'exécution (**EE3"**) qui permet de décrémenter l'état de tous les événements de l'opérande considéré dans les deux cas suivants :
   i) si la garde est évaluée à faux dès le début : aucune des itérations du fragment combiné LOOP n'est exécutée, donc nous décrémentons les états relatifs à tous les événements de l'opérande y compris ceux qui sont compris dans les opérandes qui sont imbriqués dans l'opérande considéré,
   ii) si la valeur de la garde change et devient fausse au cours des itérations : dans ce cas, les événements de l'itération courante doivent achever leurs occurrences (si nous avons des messages qui sont déjà émis, leurs événements reçus correspondants doivent se déclencher : pour un événement donné nous aurons autant de réceptions que d'émissions) et les événements des prochaines itérations ne doivent pas se déclencher.

   Considérons l'événement *evt* comme un événement fictif négatif d'un fragment combiné LOOP appartenant à l'opérande Y tel que $Y = EVT\_D^{-1}(evt)$. Pour tout événement e appartenant au même opérande Y tel que $e \in EVT\_G(Y)$, nous décrémentons son état en tenant compte de toutes les itérations à ignorer. L'événement e peut être localisé soit directement dans l'opérande Y ou il peut être localisé dans un opérande qui est imbriqué dans l'opérande Y. Soit X l'opérande qui contient directement l'événement e tel que $X = EVT\_D^{-1}(e)$. Pour empêcher l'occurrence de e pour une itération, nous décrémentons son état par rapport à son poids relativement à l'opérande Y ($weight(Y, X)$). Pour ignorer plusieurs itérations, il suffit de multiplier son état par ce poids relatif par le nombre des itérations restantes à ignorer.
   Par conséquent, nous introduisons l'effet d'exécution (**EE3"**) de l'événement *evt* qui permet d'agir sur les états de tout événement e.

$$state(e) := state(e) - ((state(evt)\ mod\ weight(Y)) * weight(Y, X)) \quad \textbf{(EE3")}$$

Finalement, pour tous les événements fictifs positifs et négatifs des fragments combinés ALT et LOOP nous rajoutons un effet d'exécution **EE4** qui permet de mettre à jour la valeur de la garde afin d'obtenir tous les comportements possibles.

Remarque : pour un fragment combiné STRICT nous avons défini pour chaque opérande un événement fictif qui ne se déclenche que si tous les événements qui sont contenus dans l'opérande considéré sont consommés. Cet événement fictif possède des conditions de déclenchement et des effets d'exécutions qui sont similaires à ceux d'un événement normal.

4.5 Récapitulatif

Nous récapitulons dans le Tableau 4.4 les conditions de déclenchement et les effets d'exécutions selon le type des événements que nous avons identifié dans un diagramme de séquence d'UML2.X équipés de FC imbriqués.

Types d'événement	**Conditions de déclenchement**	**Effets d'exécution**
Événement normal	CD1 et / ou CD1' CD2	EE1 EE2
Événement fictif positif d'un FC ALT	CD1 et / ou CD1' CD2 CD3	EE1 EE2 EE1' EE3 EE4
Événement fictif négatif d'un FC ALT	CD1 et / ou CD1' CD2 CD3	EE1 EE2 EE3' EE4
Événement fictif positif d'un FC LOOP	CD1 et / ou CD1' CD2 CD3	EE1 EE2 EE1' EE4
Événement fictif négatif d'un FC LOOP	CD1 et / ou CD1' CD2 CD3'	EE1 EE2 EE3" EE4

TABLE 4.4 – Les conditions de déclenchement et les effets d'exécutions des événements

4.6 Conclusion

La plupart des sémantiques qui ont été proposées pour les DS sont des sémantiques de traces. Elles ne sont pas très pratiques pour la compréhension et l'analyse des comportements des systèmes modélisés par les DS. De plus le calcul des traces même pour un DS basique n'est pas si évident. En effet le nombre de traces générées en considérant le chevauchement de l'occurrence des événements croit de façon factorielle. Sachant que la plupart de ces sémantiques ne proposent pas d'outils pour la génération des traces.

En outre, les sémantiques opérationnelles existantes sont basées sur les définitions du standard OMG pour le calcul des traces, par conséquent elles ne sont pas adéquates aux DS modélisant les comportements des systèmes distribués. D'autant plus que dans les travaux existants dont les sémantiques sont essentiellement compositionnelles, ce qui veut dire que les FC imbriqués sont supportés implicitement, cependant il n'est pas montré explicitement les stratégies des exécutions des événements appartenant à des FC imbriqués. En contrepartie, les approches qui traitent les DS avec des FC imbriqués émettent des hypothèses assez restrictives pour éviter les

inconsistances et faciliter les stratégies d'exécutions des événements appartenant à ces FC, ce qui limite le pouvoir d'expression de ces FC [Yo06], [Gab08], [Ale04], [Sha11].

La sémantique opérationnelle que nous avons proposée dans ce chapitre remédie aux insuffisances des sémantiques opérationnelles existantes, en effet elle est basée sur une sémantique d'ordre partiel dont ses relations tiennent compte de l'indépendance des composants modélisés dans un environnement distribué, elle traite les FC convenablement (conservation de leurs interprétations standard) et elle traite les FC imbriqués explicitement. En outre la sémantique opérationnelle, supporte les gardes explicitement. Dans la sémantique opérationnelle, nous avons défini, exprimé et formalisé de façon non ambiguë la forme de chaque type d'événement pouvant figurer dans un DS pouvant contenir notamment des FC imbriqués.
La forme d'un événement inclut ses conditions d'occurrences ainsi que les effets de son occurrence. La vérification de l'occurrence des événements précédents est l'une des contraintes importantes qui consiste à la détermination de toutes les relations de précédence de l'événement considéré. Une autre contrainte consiste à l'évaluation de la garde pour les événements appartenant à des FC contenant des gardes. Ces contraintes nécessitent un prétraitement qui est traité dans la sémantique causale.
Nous avons proposé une approche pour traiter le problème de la synchronisation entre les lignes de vie dans un contexte où les composants modélisés sont distribués. Finalement, nous avons défini le comportement d'un diagramme de séquence d'UML2.X.

La sémantique opérationnelle facilite l'implémentation du langage diagrammes de séquence d'UML2.X et son analyse. Elle servira comme base pour exprimer la relation de raffinement entre les DS qui sont utilisés pour une modélisation incrémentale des comportements des systèmes distribués par raffinement des diagrammes de séquence d'UML2.X. La sémantique opérationnelle que nous avons proposé est indépendante de tout formalisme cible et elle peut être implémentée par n'importe quel outil.

Chapitre 5

Raffinement des diagrammes de séquence UML2.X

Sommaire

5.1 Introduction au raffinement

Le raffinement est une technique qui est désormais utilisée pour gérer et dominer la complexité de construction des systèmes étudiés. Dans l'approche orientée objet, le raffinement n'est pas une notion native, et ses origines se retrouvent dans les méthodes formelles. Le raffinement a été introduit dans les années 1970 par [N. 71] : *le raffinement est défini comme une méthode de développement top-down de programme. Des abstractions qui sont décrites de façon informelle sont graduellement remplacées par des détails formels plus fins.*
Une autre interprétation du raffinement consiste à développer formellement les programmes à partir des spécifications, en préservant la correction entre les spécifications et programmes.
Cette notion est ensuite reprise par les travaux de [E. 76], puis formalisée par [Ral78] dans les années 1980. Cette notion a été développée plus tard dans les travaux de [M. 88a] et [J.-96b], les auteurs ont développé cette notion.

Selon le standard de l'OMG, le raffinement est défini comme une relation existante entre un modèle spécifié à un certain niveau de détail et une spécification enrichie. La notion de raffinement

dans UML a été initialement exploitée pour préserver la cohérence inter-diagrammes ([S. 05], [Mit03]) et intra-diagrammes ([Jea02], [Joh02], [Has03], [Ger01]).
La cohérence inter-diagrammes s'intéresse à la préservation de la cohérence entre des diagrammes hétérogènes ou distincts. En effet, dans la conception logicielle, le concepteur a souvent recours à différents diagrammes d'UML pour modéliser d'une part les vues statiques durant les phases de la spécification des exigences (diagrammes de cas d'utilisations, diagrammes de classes) et d'autre part les vues dynamiques durant les phases de conception et d'analyse (diagrammes de séquence, diagrammes de collaborations, etc...) du système. Dans les travaux de [Rag04], les auteurs se sont intéressés à l'étude de la cohérence entre les diagrammes de séquence et les machines à états de protocoles dans le contexte de la restructuration (*refactoring*) de modèles. Dans les travaux de [S. 05] ainsi que les travaux de [Gre02], les auteurs ont étudié la cohérence entre les diagrammes de séquence et les machines à états. Dans les travaux de [Rag03b], les auteurs ont étudié la cohérence entre les diagrammes de classes, les diagrammes de séquence et les machines à états de protocoles.
La cohérence intra-diagrammes consiste à la préservation des propriétés par les diagrammes étudiés qui subissent un développement en plusieurs phases afin d'obtenir des diagrammes plus détaillés.

Il existe plusieurs travaux qui se sont intéressés à la cohérence intra-diagrammes [Ger01], [Jea02], [Joh02], [Has03].

5.2 Relations de raffinement existantes et leurs propriétés

5.2.1 Préliminaires

5.2.1.1 Un système de transitions étiquetées

Un système de transitions étiquetées (*labelled transition system* (LTS)) est un système ayant des transitions qui sont étiquetées avec des actions. L'ensemble des actions d'un LTS est appelé son alphabet de communication et il constitue le support des interactions que le système peut avoir avec son environnement.

En outre, un LTS peut avoir des actions qui sont non-observables par l'environnement, elles sont appelées τ transitions.
Soit *States* l'ensemble universel des états, $Act_\tau = Act \cup \{\tau\}$ où *Act* est l'ensemble universel des étiquettes des actions observables et τ est l'étiquette des actions non observables.

Definition 14. *Un LTS est un tuple* $M = (S, L, \longrightarrow, S_0)$ *où* $S \subseteq States$ *est un ensemble fini d'états,* $L \subseteq Act_\tau$ *est un ensemble d'étiquettes,* $\longrightarrow \subseteq (S \times L \times S)$ *est une relation de transition entre les états,* $s_0 \in S$ *est l'état initial. Soit* M *un LTS tel que* $M = (S, L, \longrightarrow, s_0)$*, on appelle transition sur* l *de* M *à* M' *noté* $M \xrightarrow{l} M'$ *si* $M' = (S, L, \longrightarrow, s'_0)$ *et* $(s_0, l, s'_0) \in \longrightarrow$*.*

De façon similaire on écrit $M \xrightarrow{\hat{l}} M'$ pour désigner $M \xrightarrow{l} M'$ ou $l = \tau$ et $M = M'$. On utilise $M \stackrel{l}{\Longrightarrow} M'$ pour désigner $M(\xrightarrow{\tau})^* \xrightarrow{l} (\xrightarrow{\tau})^* M'$
et $M \stackrel{\hat{l}}{\Longrightarrow} M'$ pour désigner $M(\xrightarrow{\tau})^* \xrightarrow{\hat{l}} (\xrightarrow{\tau})^* M$

Definition 15. *Soient* s *un état et* α *une action, l'ensemble des états successeurs de* s *après l'exécution de* α *est définie comme suit*

$$Post(s, \alpha) = \{s' / s \xrightarrow{\alpha} s'\}$$

$Post(s, \alpha)$ *contient tous les états qui peuvent être atteints de l'état* s *en exécutant l'action* α*.*

$$Post(s) = \bigcup_{\alpha \in L} Post(s, \alpha)$$

Par conséquent $Post(s)$ contient tous les états atteignables de l'état s en exécutant n'importe quelle action.

5.2.1.2 Les systèmes de transitions étiquetées gardées

Nous étendons la définition des LTS en tenant compte des gardes. Ainsi, on les appelle systèmes de transitions étiquetées gardées.

Definition 16. *Un LTS gardé est un tuple $P = (S, L, \longrightarrow, S_0, G(x))$ où $L, \longrightarrow, S_0$ sont définis de la même manière que dans un LTS.*
$G(x)$ est un ensemble de prédicats qui sont définis sur des variables ou par vrai (par défaut la garde est vraie).

Une transition $(s, [g], e, s')$ peut être activée à partir de l'état s où sa garde g est vraie, dans ce cas la transition est valide. Une transition gardée est notée $s \xrightarrow{[g]e} s'$.

5.2.2 Diversité des relations de raffinement

La démarche de construction d'un modèle par raffinements successifs exige de vérifier que chaque modèle préserve bien sa correction vis-à-vis du modèle qui le précède. Ceci nécessite au préalable la formalisation puis la vérification d'une relation de raffinement. Il existe plusieurs relations de raffinement qui ne sont pas toutes équivalentes. Nous citons en l'occurrence les relations les plus répandus mais dans lesquelles les notions de raffinement ne sont pas définies de façon similaire.
• les relations d'équivalence : un système est équivalent à un autre si pour tous les états et les actions qui sont exécutés par un système correspondent des états et des actions qui sont exécutés par l'autre système. On dit que le système peut *mimer* chaque transition de l'autre système. Vus de l'extérieur, les systèmes considérés sont indiscernables.
Dans la littérature il existe plusieurs relations d'équivalence qui sont définies sur les systèmes de transitions étiquetées (*labelled transition systems*) et sur l'algèbre de processus (*process algebras CCS*)[R. 89]. Nous citons les plus répandues [Mic00] :

- les équivalence de traces (*trace equivalence*) [R. 89],
- les équivalence des échecs (*failures equivalence*) [M. 87],
- l'équivalence de bi-simulation forte (*strong bisimulation equivalence*) [R. 89] et
- l'isomorphisme (*isomorphism*).

Les relations d'équivalence sont des relations réflexives, transitives, et symétriques.
• les relations de simulations ([R. 99]),
• les relations de raffinement : il existe plusieurs relations de raffinement [R. 93] qui ont été définies sur les *LTS* et *CCS*, nous citons les plus populaires :

- le raffinement de traces (*trace refinement*) [M. 87] : c'est la relation de raffinement faible (*weak refinement relation*) qui assure que le modèle raffiné peut uniquement, mais pas nécessairement, exécuter les traces qui peuvent être aussi exécutées dans le modèle abstrait,
- le pré-ordre d'échec (*failure preorder*),
- le pré-ordre de spécification (*specification preorder*) et
- le pré-ordre de réduction double (*double reduction preorder*).

Nous retrouvons aussi d'autres relations qui ont été utilisées dans les approches de construction incrémentale.
• la relation de conformité (*conf*) : c'est une relation plus forte que la relation de raffinement. La relation de conformité a été formalisée dans les travaux de [E. 86] et [J. 99e]. Elle a été proposée

comme une relation d'implantation pour vérifier si un modèle d'une implantation respecte sa spécification dans le réseau des télécommunications,
• la relation d'extension (*ext*) et de réduction (*red*) : ce sont des variantes de la relation de conformité, auxquelles des extensions ou des réductions de traces sont combinées. Parmi les relations de réduction, il y a des relations de pré-ordre de nécessité et de pré-ordre de possibilité proposées par [M. 88b] et implantées par [Ran93].
Les relations de conformité (*conf, ext et red*) ont été utilisées par les approches de raffinement de construction incrémentale par le biais de plusieurs diagrammes. Par exemple dans les travaux de [Hon08], elles ont été exploitées pour la construction incrémentale par raffinement des machines à états. Dans les travaux de [Lu 11], la relation de conformité a été utilisée et formalisée pour un développement incrémental des systèmes par raffinement d'une sous-classe de diagrammes de séquence tout en préservant des comportements bien spécifiques (nommés les comportements requis).

Le choix d'une relation de raffinement dépend de plusieurs facteurs comme par exemple le langage utilisé, les propriétés visées sur les modèles raffinés, etc.... .

5.2.3 Définitions formelles de quelques relations de raffinement

Nous reprenons quelques définitions formelles, de la littérature [D. 09], [Rob71], des relations les plus populaires et les plus utilisées pour exprimer une relation de raffinement [D. 09].

5.2.3.1 Relations de simulation

La relation de simulation a été définie [Rob71] et formalisée pour les programmes par [MA70] : un programme P simule un programme Q si les deux font les mêmes principaux calculs. La simulation est une technique qui consiste d'une part à expliciter une relation entre un programme et son raffinement et d'autre part à vérifier que cette relation satisfait certaines propriétés.

Definition 17 (**La relation de simulation**). *Considérons deux LTS M1 et M2 tels que* $M1 = (S_1, L_1, \longrightarrow_1, S_0^1)$ *et* $M2 = (S_2, L_2, \longrightarrow_2, S_0^2)$*, soit R une relation de* S_1 *vers* S_2*. On dit que M2 simule M1 par rapport à la relation R que nous notons* $M2 \preceq_R M1$ *si et seulement si :*

⋆ $(S_0^1, S_0^2) \in R$
⋆ $(\forall e \in A)(\forall p \in S_1)(\forall q \in S_2)(\forall p' \in S_1)[(p,q) \in R \wedge p \xrightarrow{e}_1 p' \Longrightarrow (\exists q' \in S_2)[(p',q') \in R \wedge q \xrightarrow{e}_2 q']]$

Avec $L = L1 = L2$.

Les relations de simulation permettent de prouver le raffinement entre deux systèmes à l'aide de conditions suffisantes. Ces conditions sont appelées respectivement *vers l'avant (forward)* et *vers l'arrière backward*. Dans la littérature nous retrouvons les définitions relatives aux relations de simulation *forward* et *backward* sur différents formalismes : les automates [A. 96], les types de données du langage Z [J. 01], les systèmes de transitions étiquetées (LTS) [M.B88], etc... .
Nous nous inspirons des définitions qui ont été données dans [M.B88] où les auteurs cherchent des conditions suffisantes pour prouver le raffinement des processus. Et nous définissons les relations de simulations sur les systèmes de transitions étiquetées (LTS).
Ainsi les relations de simulations *forward* et *backward* sont définies sur les LTS.

5.2.3.2 Relation de simulation vers l'avant et vers l'arrière sur les LTS

Étant donné deux LTS $M1 = (S_1, L, \longrightarrow_1, S_0^1)$ et $M2 = (S_2, L, \longrightarrow_2, S_0^2)$ ayant des espaces d'états qui sont différents, et le même alphabet (même ensemble des étiquettes), nous avons soit

une relation de simulation vers l'arrière soit une relation vers l'avant qui sont définies comme suit :

1. Relation de simulation vers l'avant (*forward*) : $M1$ est dit une simulation vers l'avant de $M2$ s'il existe une relation $R_f \subseteq S1 \times S2$ telle que :

$\star\ \forall s_1 \in S_1, s_2 \in S_2 \bullet s_1\ R_f\ s_2 \Longrightarrow\ Post_1(s1) = Post_2(s2)$

$\star\ \forall s_1 \in S_1, ((s_2, s_2^{'}) \in S_2^2), \alpha \in L \bullet s_1\ R_f\ s_2 \wedge s_2 \xrightarrow{\alpha}_2 s_2^{'} \Longrightarrow (\exists s_1^{'} \in S_1 \bullet s_1 \xrightarrow{\alpha}_1 s_1^{'} \wedge s_1^{'}\ R_f\ s_2^{'})$

$\star\ \forall s_2 \in S_0^2 \bullet \exists s_1 \in S_0^1 \bullet s_1\ R_f\ s_2.$

Avec cette règle *l'avant*, le LTS $M1$ peut simuler le comportement du LTS $M2$ tant que les deux LTS sont dans des états correspondants.

2. Relation de simulation vers l'arrière (*backward*) : $M1$ est dit une simulation vers l'arrière de $M2$ s'il existe une relation $R_b \subseteq S2 \times S1$ telle que :

$\star\ \forall s_2 \in S_2 \bullet \exists s_1 \in S_1 \bullet s_2\ R_b\ s_1 \Longrightarrow\ Post_1(s_1) \subseteq Post_2(s_2)$

$\star\ \forall s_1^{'} \in S_1, ((s_2, s_2^{'}) \in S_2^2), \alpha \in L \bullet s_2^{'}\ R_b\ s_1^{'} \wedge s_2 \xrightarrow{\alpha}_2 s_2^{'} \Longrightarrow (\exists s_1 \in S_1 \bullet s_1 \xrightarrow{\alpha}_1 s_1^{'} \wedge s_2\ R_b\ s_1)$

$\star\ \forall s_1 \in S_1, s_2 \in S_0^2 \bullet\ s_2\ R_b\ s_1 \Longrightarrow s_1 \in S_0^1.$

Avec cette relation, si $M2$ atteint un certain état, alors $M1$ est capable de reproduire la séquence d'événements afin de trouver un état correspondant à partir duquel $M1$ peut simuler le comportement de $M2$: il s'agit d'une simulation *backward*.

5.2.3.3 Relation d'équivalence de bi-simulation

Definition 18 (**La relation de bi-simulation**). *soient M1 et M2 deux LTS tels que* $M1 = (S_1, L_1, \longrightarrow_1, S_0^1)$ *et* $M2 = (S_2, L_2, \longrightarrow_2, S_0^2)$ *et R une relation de* S_1 *vers* S_2. *M1 et M2 sont bi-similaires par rapport à la relation R que nous notons* $M2 \approx_R M1$ *si et seulement si :*

— $M2 \preceq_R M1$

— $M1 \preceq_{R^{-1}} M2$

1. Relation d'équivalence de bi-simulation forte (*strong bisimulation equivalence*) Soit ξ l'univers de tous les LTS. Soient $M1, M2 \in \xi$. M1 et M2 sont équivalents forts (*strong equivalent*) que nous notons $M1 \sim M2$ si $\alpha M1 = \alpha M2$, (où $\alpha M1 = L\ \tau$ et $\alpha M2 = L\ \tau$ désignent respectivement l'alphabet de communication de M1 et M2), et (M1, M2) sont contenus dans une relation de bi-simulation $R \subseteq \xi \times \xi$ pour laquelle ce qui suit se produit pour tout $l \in Act_\tau$:

1. $(M1 \xrightarrow{l} M1') \Longrightarrow (\exists M2' \bullet M2 \xrightarrow{l} M2' \wedge (M1', M2') \in R)$
2. $(M2 \xrightarrow{l} M2') \Longrightarrow (\exists M1' \bullet M1 \xrightarrow{l} M1' \wedge (M1', M2') \in R)$

Cette relation ne distingue pas τ comme des actions spéciales ou non observables.

2. Relation d'équivalence de bi-simulation faible (*weak bisimulation equivalence*) : la relation de bi-simulation faible considère des séquences d'exécutions et non des actions seulement. Elle se définit de la même façon que la relation de bi-simulation forte en remplaçant les transitions de la forme $s \xrightarrow{a}$ par des chemins $s \stackrel{a}{\Longrightarrow}$, ce qui signifie que l'état s' peut être atteint à partir de l'état s en exécutant la séquence α avec $(\alpha = a...b...c)$ et a, b et c sont des actions.

soit ξ l'univers de tous les LTS. Soient $M1, M2 \in \xi$. M1 et M2 sont équivalents faibles (*weak equivalent*) noté $M1 \approx_w M2$ si $\alpha M1 = \alpha M2$, (où $\alpha M1 = L\ \tau$ et $\alpha M2 = L\ \tau$ désignent respectivement l'alphabet de communication de M1 et M2), et (M1, M2) sont contenus dans une relation de bi-simulation faible $R \subseteq \xi \times \xi$ pour laquelle ce qui suit se produit pour tout $l \in Act_\tau$:

1. $(M1 \xrightarrow{l} M1') \Longrightarrow (\exists M2' \bullet M2 \stackrel{\hat{l}}{\Longrightarrow} M2' \wedge (M1', M2') \in R)$
2. $(M2 \xrightarrow{l} M2') \longrightarrow (\exists M1' \bullet M1 \stackrel{\hat{l}}{\Longrightarrow} M1' \wedge (M1', M2') \in R)$

Cette relation compare les comportements observables des modèles et ignore les τ transitions.

5.2.3.4 Les relations de raffinement

Le raffinement est le type le plus simple de simulation [A. 96]. Étant donné deux LTS M1 et M2 et une fonction R qui est définie des états de M1 aux états de M2. On dit qu'il existe un raffinement de M1 à M2 si les deux conditions suivantes sont satisfaites :
$*$ $s \in S_0^1$ alors $R(s) \in S_0^2$
$*$ $s' \xrightarrow{l}_1 s$ alors $R(s') \stackrel{\hat{a}}{\Longrightarrow}_2 R(s)$

Nous distinguons le raffinement fort (*strong refinement*) et faible (*weak refinement*) et ils se définissent de la manière que les relations de bi-simulation fortes et faibles.

Dans ce qui suit nous énumérons quelques propriétés des relations de raffinement.

5.2.4 Propriétés d'une relation de raffinement

Une relation de raffinement peut assurer une ou plusieurs propriétés parmi les propriétés ci-dessous. Notons que la liste n'est pas exhaustive.
• la réflexivité : cette propriété assure qu'un programme est un raffinement de lui-même,
• la transitivité et la monotonie : ce sont des propriétés importantes et essentielles. La transitivité permet d'assurer que les spécifications intermédiaires ou finales possèdent les mêmes propriétés que la spécification initiale. La monotonie sert à raffiner les parties d'un programme ou d'un modèle séparément,
• la réduction de l'indéterminisme. Un système indéterministe est un système qui produit des comportements différents à chaque exécution du système. Cette définition est inspirée de la définition du déterminisme formulée par [C. 78] et par [R. 99] : *si l'on effectue la même expérience deux fois sur un système déterminé, en commençant à chaque fois dans son état initial, alors nous nous attendons à obtenir le même résultat, ou comportement, à chaque fois*,
• la réduction de la partialité : ceci se fait en rajoutant plus de composants et de fonctionnalités,
• la réduction des traces : le modèle raffiné produit moins de traces que le modèle abstrait,
• l'extension de traces : elle correspond à l'ajout de nouvelles actions ou de nouveaux événements observables à partir d'un état initial.
Bien qu'il existe une panoplie de propriétés, une relation de raffinement doit garantir un minimum de propriétés qui sont indispensables à un développement incrémental. Ces relations sont la transitivité, la réflexivité et la monotonie. Si une relation ne garantit pas une propriété souhaitée, des manières détournées peuvent être utilisées pour l'assurer. Par exemple le raffinement de traces ne garantit pas la réduction de la partialité [Mic00]. Cependant une définition adéquate du langage de spécification permet de garantir cette propriété [B. 96].

5.2.5 Expression d'une relation de raffinement

5.2.5.1 Préliminaires

Partant d'abord de la forme basique d'un programme qui est formulée dans la logique de Hoare, un programme P est associé à une précondition S et une postcondition Q tels que la précondition S représente les états possibles avant l'exécution de P et la postcondition Q est vérifiée par les états possibles après l'exécution de P. Un programme P est correct vis-à-vis de sa spécification S et Q si à partir de tout état initial vérifiant la précondition S, le programme se termine et fait passer le système à un état satisfaisant la postcondition Q.
La correction d'un programme P par rapport à sa spécification S et Q est notée $S\ \{P\}\ Q$.
Étant donné $S\ \{P\}\ Q$, nous cherchons un programme P' tel que :

$$\forall S, Q, (S\ \{P\}\ Q \Longrightarrow S\ \{P'\}\ Q)$$

Autrement dit, toute spécification qui est satisfaite par P est aussi satisfaite par P'. Nous pouvons remplacer le programme P par P', tout en continuant à satisfaire la spécification initiale. P' est un raffinement de P, noté par :

$$P \sqsubseteq P'$$

Traditionnellement il faut exprimer une relation de simulation entre le programme P et P', ce qui revient à exprimer des conditions suffisantes sur la relation pour assurer la correction du raffinement. Ces conditions dépendent étroitement de la sémantique utilisée pour interpréter $S\{P\}Q$. Dans [Fré05], les auteurs ont précisé que les conditions de raffinement constituent un problème de sémantique. En effet les différentes sémantiques existantes n'ont ni la même abstraction ni la même vision des programmes. Les auteurs présentent un état de l'art sur quelques sémantiques existantes en explicitant les conditions d'un raffinement correct pour chacune :

• la sémantique de transformateurs de prédicats,
• la sémantique des jeux,
• la sémantique du choix,
• la sémantique relationnelle,
• la sémantique opérationnelle avec les (LTS) et
• la sémantique dénotationnelle (trace-divergence).
Dans les trois premières sémantiques, des paramètres qui assurent la correction des programmes sont d'abord fixés. Ces paramètres sont respectivement basés sur :

• la notion de *plus faible précondition*, notée *wp*, qui est basée sur la logique de *Hoare*. L'opérateur *wp* permet de calculer la plus faible précondition qui garantisse la terminaison d'un programme P et qui laisse le système dans un état qui satisfait la postcondition Q. Pour un programme P et une postcondition $Q, wp(P, Q)$ est la plus faible précondition S telle que $S\ \{P\}\ Q$. La correction et le raffinement sont maintenant exprimés dans cette sémantique.
Un programme S de précondition P et de postcondition Q est correct si :

$$S \Longrightarrow wp(P, Q)$$

Dans la sémantique *wp*, le raffinement se traduit par :

$$P \sqsubseteq P' \Longleftrightarrow (\forall Q, wp(P, Q) \Longrightarrow wp(P', Q))$$

À partir d'un programme P et d'une postcondition Q, l'opérateur *wp* permet de revenir sur les préconditions possibles. La sémantique *wp* est dite *backward*,
• la notion de contrat qui a été introduite par Back [R.J98]. Un contrat C est une généralisation des programmes et des spécifications ; il est déterminé par un langage sur les instructions qui sont composées par des séquences et des choix.

Une instruction peut avoir une forme simple (affectation d'une valeur) comme elle peut être composée d'assertions (représentant des conditions) et des hypothèses (représentant des assomptions du contrat),
• à l'inverse de la sémantique des transformateurs de prédicats qui repose sur la sémantique *wp*, la sémantique du choix est une sémantique *forward*.

La sémantique *wp* constitue la base des raffinements dans les méthodes formelles Z [J. 96a], les systèmes d'actions (*actions systems*)[R.J98], les processus séquentiels communicants (CSP) [C. 78], en particulier B [J.-96b] et B événementiel [J.-10].

Le raffinement entre les programmes s'exprime à différents niveaux selon la sémantique considérée. Par exemple dans la sémantique relationnelle, le raffinement s'exprime au niveau des types de données, dans la sémantique opérationnelle il s'exprime au niveau des transitions et dans la sémantique dénotationnelle il s'exprime au niveau des traces.

Bien que certaines relations (telles que les relations conf, ext et red) ont été prouvées mathématiquement, leurs applications pratiques s'avèrent difficiles, d'où le recours aux approches formelles supportant le concept de raffinement et offrant des outils supports facilitant leurs implémentations et leurs vérifications sur des exemples concrets. Citons quelques travaux autour du langage UML qui ont eu recours à cette méthodologie.
Dans [J. 99c] [J. 99b], les auteurs ont traduit la structure des machines à états UML vers un système de réécriture en langage PROMELA.
Dans [Ste99], les auteurs ont exprimé les machines à états vers la logique temporelle arborescente.
Dans [Tim01], [Ale02], [A. 02b], les auteurs ont proposé des approches pour traduire les machines à états en langage PROMELA et les diagrammes de collaboration en des ensembles d'automates de Büchi.
Dans [Sve04], les auteurs ont présenté une approche de développement incrémentale des modèles UML/RT. Ils ont procédé à la transformation des modèles UML/RT en des modèles logiques de l'environnement HUppaal [A. 02a].
Dans [S. 05], les diagrammes de séquence et les machines à états sont traduits dans l'algèbre de processus π-calcul.
Dans [Rag03a], les auteurs proposent la transformation des diagrammes de séquence, des diagrammes de classes et des machines à états dans la logique de description.

5.2.5.2 Qu'est ce qu'on exprime dans une relation de raffinement : description informelle des conditions basiques de raffinement

Nous décrivons de façon informelle les conditions basiques sur l'initialisation et les opérations. Pour l'initialisation nous avons la condition suivante : pour tout état initial concret, il existe un état initial abstrait. Pour les opérations [J. 01], [A. 14] nous avons les conditions suivantes :

1. **cohérence** (*consistency*) : l'effet de l'opération concrète correspond à un effet qui est permit par l'opération abstraite dans un état correspondant **(1)**,
2. **cohérence restreinte** (*restricted consistency*) : dans les états où l'opération abstraite est habilitée (autorisée) dans un état correspondant, l'effet de l'opération concrète correspond à un effet qui est permit par l'opération abstraite dans un tel état **(2)**,
3. **déclenchabilité** (*enabledness*) : lorsque les opérations peuvent être évoquées dans un état abstrait, elles peuvent être également évoquées dans l'état concret correspondant (dans le but de comparer les effets des opérations concrètes et abstraites) **(3)** .

La première et la deuxième condition représentent les conditions essentielles du raffinement. La deuxième condition est une version plus faible de la première : c'est une simple implication de la première condition. Ces conditions sont incorporées de différentes façons dans les relations de raffinement existantes. Une relation de raffinement peut satisfaire une ou plusieurs conditions. Par exemple le raffinement de traces est caractérisé uniquement par la satisfaction de la première condition. La relation de raffinement de B événementiel satisfait conjointement la première et la troisième condition.

5.2.6 Vérification d'une relation de raffinement

La plupart des travaux existants exploitent les techniques des explorateurs de modèles (*model checker*) offertes par les méthodes formelles pour la vérification et la correction d'une relation de raffinement. Reprenons les travaux que nous avons susmentionnés. Dans les travaux de [J. 99c], [J. 99b], les auteurs ont utilisé l'outil d'explorateur de modèles *SPIN*. Dans les travaux de [Ste99], les auteurs ont utilisé l'environnement JACK [Ama94]. Dans les travaux de [Tim01], [Ale02], [A. 02b], les auteurs ont utilisé l'outil *UPPAAL*. Dans les travaux de [Sve04], les auteurs ont vérifié les modèles raffinés UML/RT par l'explorateur de modèles en utilisant l'outil *FUJABA*. La méthode formelle B événementiel génère des obligations de preuve spécifiques à la relation

de raffinement [J.-07b]. Les obligations de preuve sont déchargées par un prouver Rodin. Dans Rodin, nous retrouvons aussi l'explorateur de modèles ProB qui peut être utilisé pour la vérification de raffinement des modèles.
Certaines méthodes formelles utilisent des relations de simulation pour assurer la vérification du raffinement. Par exemple les processus séquentiels communicants (*communicating sequential processes CSP*) [C. 78], le calcul des systèmes de communication (*calculus of communicating systems CCS*) [R. 82] et l'algèbre des processus communicants (*algebra of communicating processes ACP*) [J.A85].

5.3 Étude du raffinement des diagrammes de séquence d'UML2.X

Les diagrammes de séquence modélisent la dynamique d'un système. Initialement, un diagramme de séquence est partiel, il ne modélise ni toutes les fonctionnalités souhaitées ni tous les composants du système étudié. L'application d'une démarche incrémentale à un diagramme de séquence consiste à considérer initialement un diagramme de séquence qui est partiel, puis de faire évoluer ce diagramme de séquence en apportant des modifications pour qu'il devienne plus défini et plus précis. Les modifications apportées sur un diagramme de séquence peuvent être effectuées tout en restant dans le même niveau d'abstraction comme il peuvent le transférer d'un niveau d'abstraction à un autre niveau plus concret (voir Figure 5.1).

Nous distinguons deux types de modifications ou de raffinement :
— le raffinement structurel qui concerne le changement dans la structure du diagramme de séquence considéré, c'est à dire les aspects syntaxiques du diagramme de séquence tels que les lignes de vie, les messages, les événements et les fragments combinés,
— le raffinement sémantique correspond au raffinement des traces du diagramme de séquence considéré, ceci revient à la recherche d'une relation qui exprime le lien entre les traces abstraites et les traces raffinées.

Dans ce qui suit nous dressons un bref aperçu sur le peu de travaux existants qui ont traité le raffinement des diagrammes de séquence et nous soulignons les insuffisances de leurs approches proposées.

5.3.1 Travaux connexes des raffinements des diagrammes de séquence d'UML2.X

Dans la littérature, nous retrouvons peu de travaux qui abordent le raffinement des DS ou qui exploitent les DS dans un développement incrémentale. Deux aspects pour le raffinement des DS doivent être considérés : l'aspect structurel qui inclut les différents composants d'un DS et l'aspect sémantique qui concerne les traces. Quoique les approches existantes se focalisent sur un aspect au détriment de l'autre. En outre, même en traitant un aspect des hypothèses assez restrictives sont également émises.

Dans [Ohl06], les auteurs traitent le raffinement structurel des diagrammes de séquence en définissant plusieurs lois structurelles garantissant un raffinement correct. Ils imposent dans leurs approches que les DS considérés possèdent le même alphabet. La relation de raffinement sémantique ainsi vérifiée entre les DS est une simple relation d'équivalence de traces. Dans les lois structurelles, les auteurs imposent que le DS abstrait doit avoir la même structure que le DS raffiné. Par exemple ils ne tolèrent pas l'introduction de FC dans le DS raffiné à moins qu'il ne soit déjà présent dans le DS abstrait. Pour le raffinement de messages, ils évoquent uniquement la possibilité d'ajout de nouveaux messages qui doivent impérativement être échangés entre les nouvelles lignes de vie introduites. Ces hypothèses restrictives ne sont pas pratiques : elles sont contraignantes pour le concepteur et elles ne s'appliquent que dans des cas particuliers d'exemples concrets.

Pour les autres travaux existants qui s'intéressent au raffinement sémantique (recherche d'une relation entre les traces du DS abstrait et celles du DS raffiné), leurs approches nécessitent un

lourd traitement qui est non trivial consistant à générer tous les ensembles des traces possibles positives (ou valides) et négatives (invalides) des DS abstrait et raffiné. D'autant plus que cette tâche n'est pas automatisée.
Ensuite ils recherchent une relation d'inclusion de traces entre les ensembles des traces abstraites et raffinées. Par exemple dans [R. 05], les auteurs fournissent une sémantique pour les DS en les traduisant vers les automates de Büchi ; ils génèrent pour chaque DS deux automates tels que le premier automate modélise les comportements négatives et il permet de vérifier les propriétés de sûreté de fonctionnement et le deuxième automate modélise les comportements positives et il permet de vérifier les propriétés de vivacité.

Le raffinement est défini comme une réduction des traces possibles abstraites et il est vérifié comme une relation d'inclusion de traces.

Dans [Dav08], les auteurs proposent en plus des deux catégories standard de traces valides et invalides qui désignent respectivement les traces légales et interdites, une nouvelle catégorie de traces nommée traces *contingents* pour désigner les traces inutiles. Pour le raffinement, ils le définissent comme étant la conjonction de deux relations *enrichissement* et *réduction* deux relations, telle que la première relation permet d'augmenter le nombre de traces valides et la deuxième relation permet d'augmenter le nombre de traces invalides.

Dans les travaux de [Mar04], qui sont inspirés des travaux de [Øy05], les auteurs proposent également en plus des traces valides et invalides une catégorie de traces appelée traces *non-conclusives* pour désigner les traces générées à partir de comportements qui ne sont pas modélisés (ces traces peuvent être valides ou invalides). Il définissent une relation de raffinement assez restrictive qui permet de garder toutes les traces positives dans le diagramme de séquence raffiné et les traces négatives restent les mêmes que ceux dans le diagramme de séquence abstrait. Généralement, les relations de raffinement utilisées dans ces approches sont l'inclusion de traces classique, la simulation et l'équivalence de traces.
Bien que ces approches sont très utiles pour la vérification des propriétés de sûreté et de vivacité des systèmes étudiés. Cependant elles souffrent de quelques limites.

Dans le travail de [Lu 11], les auteurs montrent les limites de ces approches qui se contentent de vérifier une relation d'inclusion de trace classique entre les traces raffinées et abstraites.

En effet ces approches ne font pas la distinction entre les traces possibles et obligatoires ; le risque qui en découle est qu'ils peuvent prouver l'existence d'une relation d'inclusion de trace entre les traces raffinées et les traces abstraites bien que certaines traces qui sont obligatoires peuvent être perdues lors du raffinement.
Rappelons que les traces obligatoires (nommées aussi traces requises) sont définies par les travaux existants [Øy05], [Lu 11] de manières différentes par rapport à la définition standard de l'OMG [Obj15]. En effet, selon la sémantique standard, les traces requises sont capturées par l'opérateur ASSERT.
Dans [Øy05], les auteurs définissent un nouvel opérateur XALT pour distinguer entre les comportements qui modélisent des choix non déterministes souhaitables qui doivent être préservés en raffinant un diagramme de séquence, tandis que les choix qui sont modélisés par l'opérateur ALT peuvent être perdus lors du raffinement.
Dans l'approche de [Lu 11] les comportements requis sont exprimés par l'opérateur ALT. Donc toutes les alternatives (ou les branches du FC ALT) ne doivent pas être perdues lors du raffinement du diagramme de séquence.

Dans l'approche de [Øy05], les auteurs catégorisent les traces en des traces positives, négatives et non-conclusives. Ils procèdent par le calcul de toutes les traces possibles des diagrammes de séquence sur lesquels ils veulent vérifier la relation de raffinement. Ensuite ils trient les traces en des ensembles de traces positives et négatives et non-conclusives. Puis ils classent les ensembles des traces possibles (positives et négatives) en des ensembles obligations. Les traces d'une même obligation servent au même but général. Une implémentation qui satisfait la spécification doit réaliser chaque obligation. Ils définissent trois types de raffinement possibles pour le système :

• l'enrichissement : consiste à réduire les traces non-conclusives en les classant soit dans l'ensemble des traces positives soit dans l'ensemble des traces négatives,
• la restriction : consiste à réduire les traces positives en déplaçant les traces perdues dans l'ensemble des traces négatives. Quant aux traces négatives et non-conclusives, elles restent les mêmes que celles qui figurent dans le diagramme de séquence abstrait,
• le détail : permet de réduire la granularité du diagramme de séquence en ajoutant certains détails.

La relation de raffinement est une inclusion de traces en préservant le comportement requis dans le DS raffiné.
Dans l'approche de [Lu 11], le raffinement est défini comme une simulation de traces en préservant le comportement requis dans le DS raffiné.

Il est à noter également que la plupart des approches existantes qui ont traité le raffinement des diagrammes de séquence ignorent la garde sauf l'approche de [Lu 11]. En effet dans leurs travaux [Lu 11], les auteurs proposent d'étendre la définition standard de trace pour tenir compte des gardes. Cependant cette nouvelle définition n'a pas été adoptée par le standard OMG d'autant plus que cette définition de la trace n'est pas intuitive (une trace est composée de l'occurrence des événements). Dans [Øy05], les auteurs considèrent uniquement les contraintes temporelles et ils procèdent à la catégorisation des traces selon les conditions.

Pour résumer, selon notre point de vue, les approches qui sont basées sur les sémantiques de traces sont compliquées puisqu'elles nécessitent un prétraitement consistant à calculer, dans un premier temps, toutes les traces des DS sur lesquels ils veulent vérifier le raffinement, puis ils les catégorisent selon les objectifs ciblés pour la vérification de la relation de raffinement. De plus la vérification des conditions des gardes est assez délicate dans ces approches.
Pour remédier aux insuffisances des travaux existants, nous proposons une approche sans émettre des hypothèses restrictives pour les règles du raffinement structurel en considérant les divers types possibles de raffinement des composants du DS. Pour le raffinement sémantique nous nous basons sur une approche intuitive en ramenant le problème à la recherche d'une relation de simulation entre les sémantiques opérationnelles des DS considérés, ce qui revient à exprimer cette relation tout simplement sur les systèmes de transitions étiquetées (LTS) étant donné qu'une sémantique opérationnelle est concrètement décrite comme un LTS.

Dans ce qui suit nous explicitons le raffinement des diagrammes de séquence que nous considérons dans notre approche, ensuite nous formalisons notre relation de raffinement.

5.3.2 Définitions informelles du raffinement des diagrammes de séquence

5.3.2.1 Raffinement structurel

Ce raffinement concerne le changement dans la structure du diagramme de séquence, c'est à dire ses aspects syntaxiques tels que les lignes de vie, les messages, les événements et les fragments combinés.

5.3.2.1.1 Raffinement de lignes de vie. Le raffinement des lignes de vie consiste soit au renommage des lignes de vie abstraites dans le diagramme de séquence raffiné, soit à la modification les lignes de vie. Le renommage des lignes de vie est appliqué dans le contexte de réutilisation ou d'instanciation des diagrammes de séquence. La modification des lignes de vie peut se faire de deux manières :

• les lignes de vie peuvent être substituées par deux ou plusieurs lignes de vie, permettant ainsi de transférer le diagramme de séquence d'un niveau d'abstraction à un autre niveau plus concret. Ce type de raffinement est appelé *raffinement vertical*,

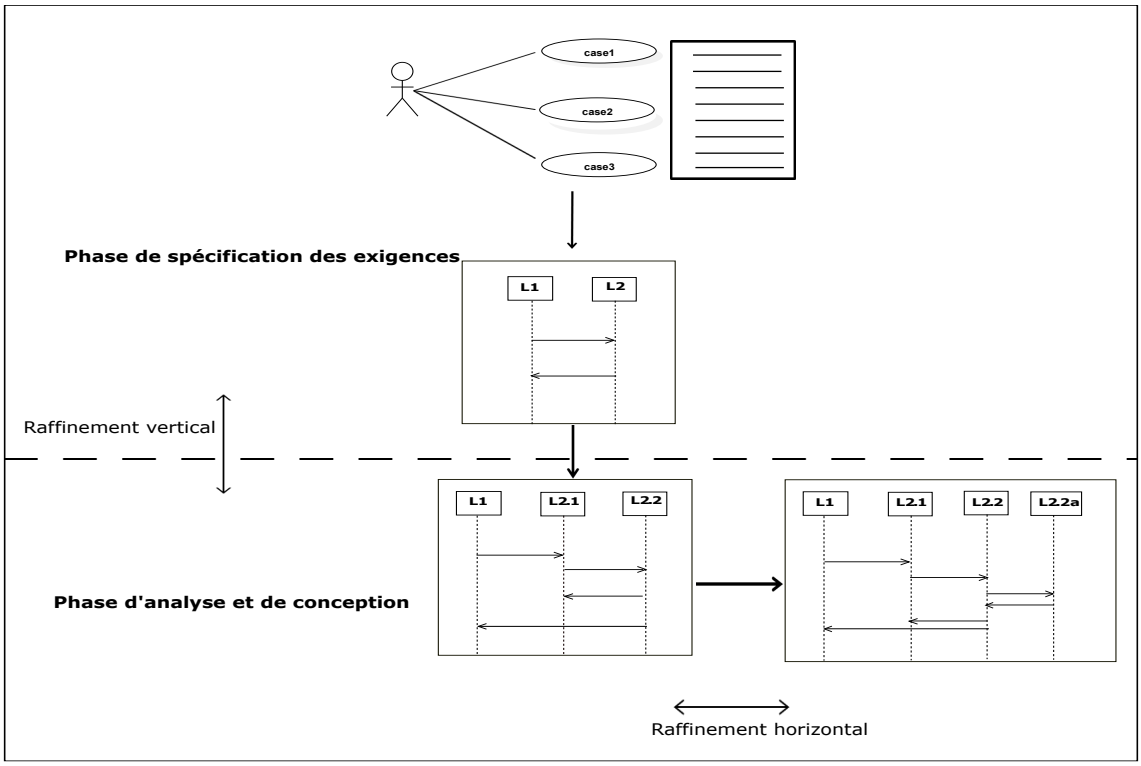

FIGURE 5.1 – Les raffinements possibles des diagrammes de séquence

• le comportement interne d'une ou plusieurs lignes de vie peut être détaillé tout en restant dans le même niveau d'abstraction. Des nouvelles sous lignes de vie sont ainsi rajoutées pour assurer quelques tâches qui sont déléguées par la ligne de vie abstraite concernée. Ce type de raffinement est appelé *raffinement horizontal*. Pour une meilleure distinction, nous appelons la ligne de vie à raffiner la ligne de vie *mère* et les nouvelles sous-lignes de vie rajoutées les lignes de vie *filles*. Une démarche de construction incrémentale peut combiner les deux types de raffinement.

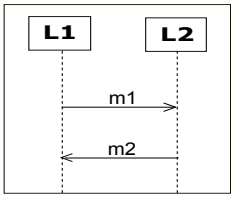

FIGURE 5.2 – DS0 abstrait

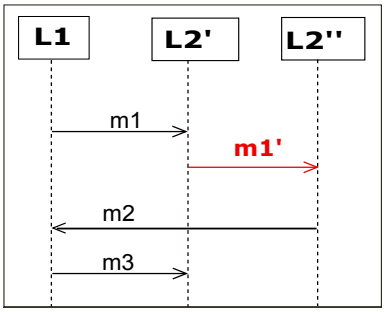

FIGURE 5.3 – DS1 : raffinement vertical du DS0

Illustration : la Figure 5.3 illustre un raffinement vertical du $DS0$ qui est représenté par la Figure 5.2 ; la ligne de vie $L2$ du $DS0$ est substituée par les lignes de vie $L2'$ et $L2''$ dans $DS1$.

La Figure 5.4 illustre un raffinement horizontal du $DS0$ qui est représenté par la Figure 5.2 ; de nouvelles lignes de vie filles $L2.1$ et $L2.2$ de la ligne de vie mère $L2$ sont rajoutées dans $DS1$.

• nous pouvons également rajouté de nouvelles lignes de vie qui sont indépendantes des lignes

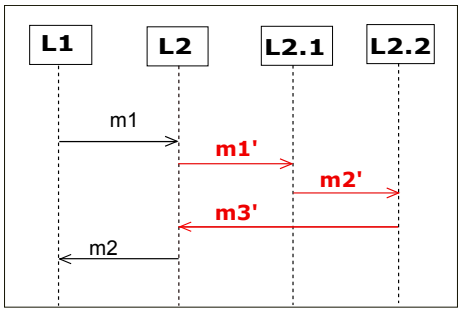

FIGURE 5.4 – DS2 : raffinement horizontal du DS0

de vie existantes.

Illustration : reprenons le DS (Figure 3.24) qui modélise les interactions au sein d'un système de gestion d'une base de données que nous avons décrit dans le Chapitre 3. Nous pouvons raffiner le DS en rajoutant une nouvelle ligne de vie administrateur *Admin* qui va interagir avec les lignes de vie existantes. Le DS raffiné est représenté par la Figure 5.5.

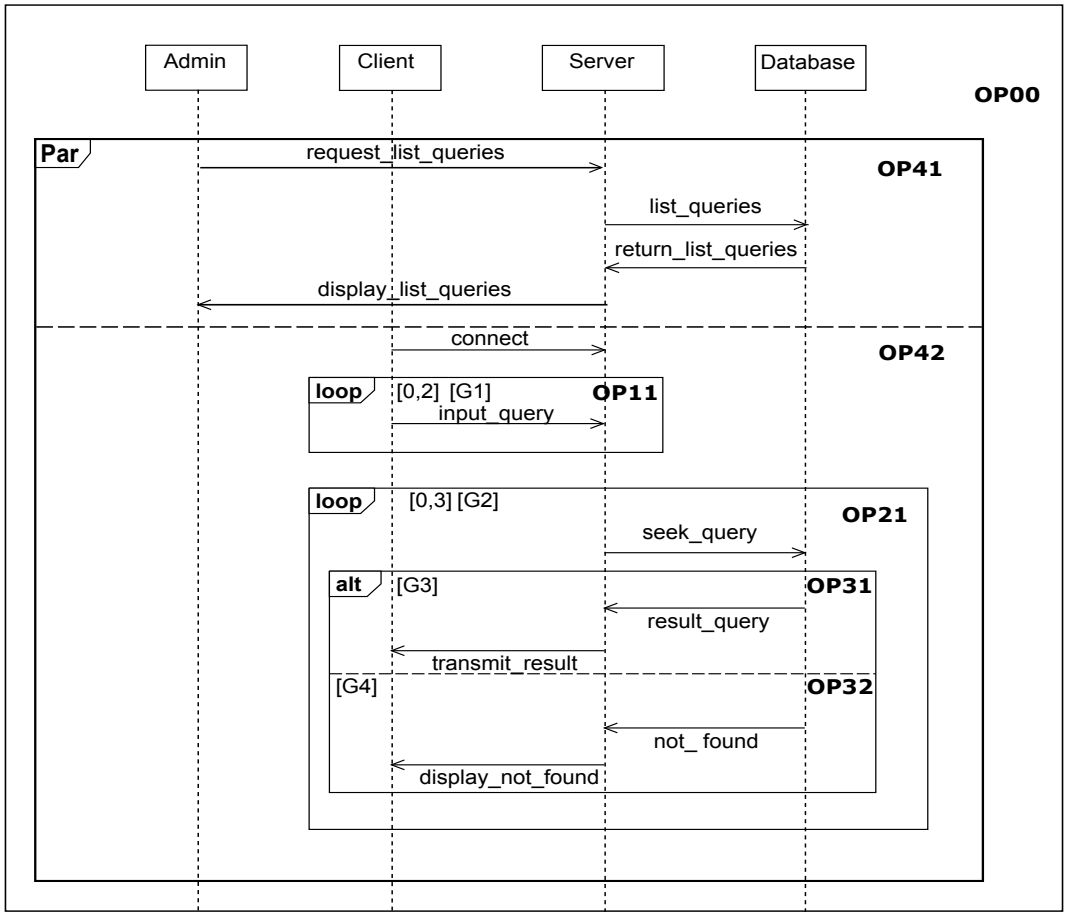

FIGURE 5.5 – Raffinement du DS de la Figure 3.24

5.3.2.1.2 Raffinement des messages et des événements. Un message est constitué d'un événement émis et d'un événement reçu. Par conséquent le raffinement des événements est lié au raffinement des messages. En raffinant un DS, soit de nouvelles interactions peuvent être ajoutées soit des interactions existantes peuvent être raffinées.

• les nouvelles interactions résultent de l'ajout de nouveaux messages,

• le raffinement des interactions existantes correspond soit au renommage des messages abstraits (ou des événements abstraits) dans le diagramme de séquence raffiné, soit à la décomposition des messages abstraits (ou des événements abstraits) ou encore à une perte de quelques messages abstraits dans le DS raffiné.
Dans la méthode B événementiel, nous retrouvons également la notion de fusion (*merging existing events*) des événements abstraits qui est définie comme une méthode possible de raffinement des événements [ROD07]. La fusion des événements abstraits en un seul événement concret se fait uniquement si certaines conditions sont satisfaites.
La décomposition des messages abstraits correspond à la notion de raffinement des actions dans les algèbres de processus [L. 92] ou au raffinement non atomique dans le langage Z (*non-atomic refinement* [J. 99a]) ou à la décomposition d'événements dans le B événementiel (*decomposing atomicity* [But09]).
Dans ses travaux [But09], l'auteur présente le raffinement d'un événement par décomposition sous la forme d'un arbre (Figure 5.6). Cette représentation est basée sur les diagramme de structures proposés par Jackson (*structure diagrams by Jackson* [Jac83]). Le nœud de l'arbre est l'événement abstrait et la descendance est composée d'un ou plusieurs événements tels que nous avons au plus un événement qui raffine l'événement abstrait (evt3 sur la Figure 5.6) et les autres événements sont des nouveaux événements qui sont supposés être des événements non visibles par l'environnement (evt1 et evt2 sur la Figure 5.6). Il impose une contrainte qui consiste à ce que l'événement qui raffine l'événement abstrait ne peut se déclencher que si tous les nouveaux événements se sont exécutés.

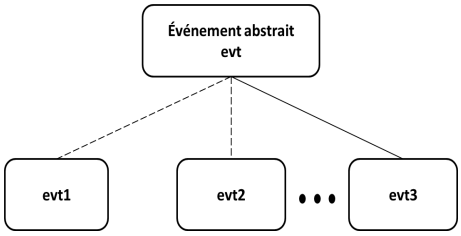

FIGURE 5.6 – Diagramme de raffinement d'événements illustrant la décomposition de l'atomicité

5.3.2.2 Raffinement des fragments combinés

Dans les travaux qui ont traité le raffinement des diagrammes de séquence d'UML2.X [Øy05], [Lu 11], le raffinement des fragments combinés n'a pas été évoqué explicitement. Considérons le sous ensemble de fragments combinés (ALT, LOOP, OPT). Intuitivement le raffinement d'un fragment combiné ALT ou LOOP ou OPT peut être défini comme sa substitution par un ou plusieurs fragments combinés, tels que nous avons entre le DS abstrait et concret une relation de raffinement (une équivalente de traces ou une inclusion des traces raffinées dans les traces abstraites).
Dans les illustrations que nous présentons ci-dessous, nous supposons que le DS abstrait et le DS raffiné possèdent le même alphabet et le même nombre de message.
• Par exemple supposons que nous avons un DS abstrait qui contient deux fragments combinés OPT (Figure 5.10), ce DS peut être raffiné par un DS comprenant un fragment combiné ALT ayant deux opérandes (ou deux branches) (Figure 5.11).
• un deuxième cas de DS abstrait qui contient un fragment combiné LOOP ayant un nombre d'itérations égale à 4 (Figure 5.7) et qui est raffiné par un DS ayant un fragment combiné avec un nombre d'itérations égale à 2 (Figure 5.8), ou encore par un DS comprenant deux LOOP imbriqués dont chaque LOOP possède un nombre d'itération qui est égale à 2 (Figure 5.9).

Dans le premier cas le deuxième cas de raffinement, nous avons à *priori* une relation de

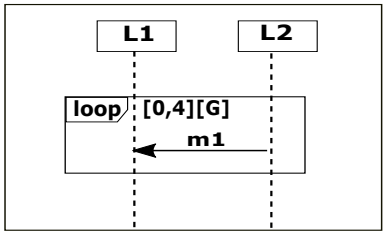

FIGURE 5.7 – DS abstrait

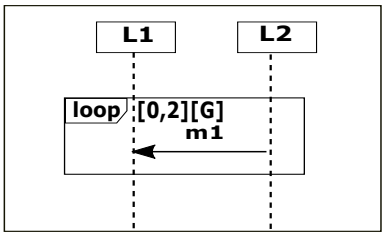

FIGURE 5.8 – Raffinement du DS de la Figure 5.7

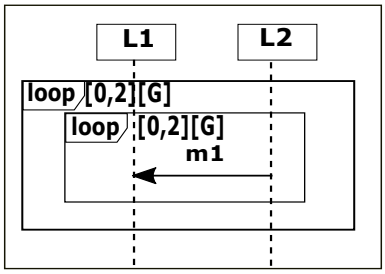

FIGURE 5.9 – Raffinement du DS de la Figure 5.7

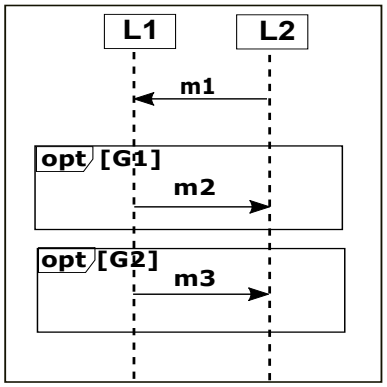

FIGURE 5.10 – DS abstrait

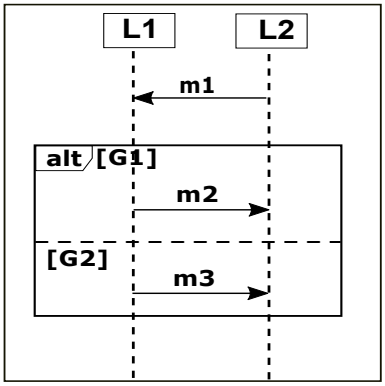

FIGURE 5.11 – Raffinement du DS de la Figure 5.10

raffinement puisque nous avons bien une inclusion de traces raffinées dans les traces abstraites. Cette conclusion reste à confirmer ou à infirmer par les expérimentations.

5.4 Notre relation de raffinement

5.4.1 Règles de raffinement

Tout d'abord nous fixons les règles d'un raffinement structurel et sémantique que nous considérons. Puis nous formalisons ces règles par la suite. Les règles sont dépendantes et elles sont étiquetées avec le symbole **Ri**.

Considérons deux diagrammes de séquence $DS1$ et $DS2$ dont $DS1$ est le diagramme abstrait et $DS2$ est le diagramme concret. Dans le diagramme de séquence $DS2$, nous raffinons une ou plusieurs lignes de vie : une ligne de vie du $DS1$ peut être soit substituée par de nouvelles lignes de vie ; soit de nouvelles lignes de vie qui y sont dépendantes peuvent être ajoutées afin d'accomplir certaines tâches qui leur sont déléguées. Les messages et les événements associés peuvent être raffinés par décomposition. De nouveaux messages (événements) peuvent être ajoutés.

Dans ces cas de raffinement possibles d'un DS donné, ici $DS1$, nous énonçons les règles suivantes qui doivent être respectées afin de confirmer que DS2 est un raffinement correct de

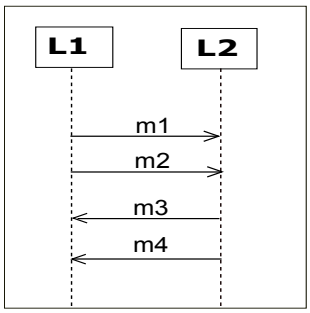

FIGURE 5.12 – DS0 abstrait

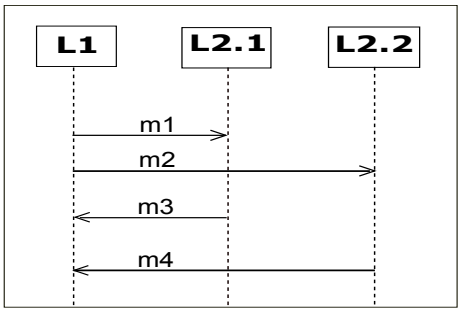

FIGURE 5.13 – DS1 : raffinement du DS0 de la Figure 5.12

$DS1$ (nous désignons par raffinement correct l'existence d'une relation de raffinement entre le DS abstrait et le DS concret). Notons que certaines règles sont spécifiques à chaque type de raffinement et d'autres sont applicables pour les deux types de raffinement. Nous illustrons ceci dans le Tableau 5.1.

• **R1** Les événements des lignes de vie abstraites substituées doivent être hérités par les nouvelles lignes de vie (i.e les événements abstraits doivent être distribués entre les nouvelles lignes de vie).

Pour le premier et le deuxième cas de raffinement, nous devons avoir les règles suivantes :

• **R2** Les lignes de vie qui ne sont pas raffinées doivent garder les mêmes événements.

• **R3** S'ils existent de nouveaux événements, ils doivent être échangés uniquement entre les nouvelles lignes de vie.

• **R4** Les nouvelles lignes de vie (filles) doivent échanger de nouveaux messages exclusivement avec la ligne de vie parente, et la ligne de vie parente doit garder les mêmes événements abstraits (R2).

• **R5** Les nouveaux événements ne doivent pas diverger pour ne pas prendre le contrôle indéfiniment et créer des blocages.

Le raffinement sémantique d'un diagramme de séquence abstrait que nous supportons consiste à la réduction de l'ensemble des traces possibles valides, de manière similaire au type de raffinement "restriction" de l'approche des Stairs [Øy05]. Nous choisissons de classer les traces perdues lors du raffinement comme étant des traces invalides. La réduction des quelques traces possibles peut se faire par exemple en changeant l'ordre visuel d'apparition de certains messages abstraits dans le DS raffiné. Ceci est permis dans notre approche grâce aux règles de la sémantique causale étant donné que les événements ne sont plus ordonnés tout au long des lignes de vie.

Illustration : considérons le DS abstrait de la Figure 5.12 ; en raffinant la ligne de vie $L2$ qui est substituée par les lignes de vie $L2.1$ et $L2.2$ (Figure 5.13), les événements qui appartiennent à la ligne de vie $L2$ vont être distribués entre $L2.1$ et $L2.2$. Rien qu'en changeant l'ordre d'apparition des messages du $DS1$ (Figure 5.14) nous réduisons le nombre de traces possibles sans altérer les règles de causalité, c'est à dire dans les deux DS ($DS1$ et $DS2$), nous avons les mêmes contraintes de causalités :

$$< sync = \{(!m1, ?m1), (!m2, ?m2), (!m3, ?m3), (!m4, ?m4)\}$$
$$< EE = \{(!m1, !m2)\}$$
$$< RE = \{(?m1, !m3), (?m2, !m4)\}$$

• Réduction du non-déterminisme pour les fragments combinés gardés : ceci est accompli en renforçant les gardes avec des conditions supplémentaires (C).

• Couvrir quelques messages abstraits avec les FC qui ont un impact sur le calcul des traces tels que le fragment combiné STRICT ou le fragment combiné ASSERT.

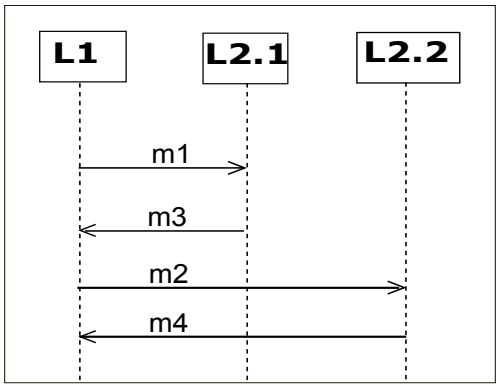

FIGURE 5.14 – DS2 : raffinement du DS1

• **R6** Tous les événements du diagramme abstrait doivent être présents dans le diagramme de séquence raffiné à un renommage près.
Pour préserver les comportements requis qui sont modélisés par l'opérateur ALT, nous gardons les branches du fragment combiné ALT (en appliquant la règle R6). Chaque trace raffinée doit avoir sa contrepartie ou son homologue dans le diagramme de séquence abstrait, en faisant abstraction des nouveaux événements introduits. Quelques approches [Ing00] proposent la suppression de quelques branches du fragment combiné ALT pour réduire le non-déterminisme. Or le non-déterminisme reflète des choix souhaitables qui doivent être préservés dans le diagramme de séquence raffiné ; cependant il peut être réduit uniquement en renforçant les gardes des différentes opérandes.

• **R7** Cette règle concerne les conditions que nous définissons pour la décomposition des messages et des événements.

Si nous avons des événements qui sont décomposés, ils doivent appartenir à la même ligne de vie de l'événement abstrait correspondant. Si la ligne de vie est décomposée alors les événements décomposés doivent être distribués entre les nouvelles lignes de vie qui raffinent la ligne de vie abstraite correspondante.

Chaque événement possède des relations de précédence avec les événements du DS. Pour les événements d'émissions : le premier événement d'émission raffiné hérite les relations de précédence de l'événement d'émission abstrait ; les événements d'émission sont liés par la relation de précédence $<_{EE}$.

Pour les événements de réception décomposés : puisque dans un contexte de composants distribués, nous n'avons aucun contrôle sur les réceptions des événements, par conséquent entre les événements de réception raffinés il n'existe aucune relation de précédence, cependant l'événement abstrait raffiné va attendre la réception des événements décomposés. Les autres événements sont considérés comme des nouveaux événements. Dans l'exemple de la Figure 5.15, les événements $!m1.1$ et $!m1.2$ sont considérés comme des nouveaux événements.

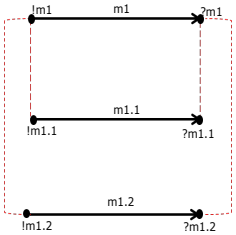

FIGURE 5.15 – Raffinement des événements par décomposition

Un message m peut être décomposé en deux ou plusieurs messages finis $m1.1$, $m1.2$,..., $m1.n$.

Soit la Figure 5.15, supposons que nous avons le message $m1$ qui est décomposé en deux messages $m1.1$ et $m1.2$. L'événement d'émission ($!m1$) et de réception ($?m1$) qui sont associés au message $m1$ sont également décomposés respectivement en $!m1.1$ et $!m1.2$ pour l'événement d'émission, et $?m1.1$ et $?m1.2$ pour l'événement de réception. Dans l'exemple de la Figure 5.15, l'événement $!m1.1$ raffiné est un nouveau événement et l'événement $!m1.2$ raffine l'événement abstrait $!m1$. Pour les relations de précédence : l'événement $!m1.1$ hérite les relations de précédence de l'événement abstrait $!m1$ et les événements décomposés $!m1.1$ et $!m1.2$ sont liés par la relation de précédence $<_{EE}$.

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**
Règles applicables au raffinement vertical	*	*	*		*	*	*
Règles applicables au raffinement horizontal		*	*	*	*	*	*

TABLE 5.1 – Spécification des règles de raffinement selon le raffinement considéré

5.4.2 Formalisation de la relation de raffinement

Considérons deux diagrammes de séquences $DS1$ et $DS2$ tels que :

$$DSi = \langle L_i, M_i, EVT_i, FCT\_s_i, FCT\_r_i, FCT\_l_i, OP_i, F_i, <_{causi}, tree\_OPi \rangle \text{ avec } i \in \{1, 2\}$$

et leurs sémantiques opérationnelles associées

$$Sem(DS_i) = \langle S_i, S_i^0, \longrightarrow_i \rangle \text{ avec } i \in \{1, 2\}$$

Raffinement structurel. Nous définissons une relation de correspondance entre les diagrammes de séquence considérés. Ceci nécessite l'intervention d'un expert. Une relation de correspondance entre les diagrammes de séquence abstrait et concret $DS1$ et $DS2$ est définie comme une correspondance entre leurs composants. Une ligne de vie L peut être soit substituée par des lignes de vie ou son comportement peut être détaillé en ajoutant des lignes de vie filles.
Par conséquent :
• une relation de correspondance doit être définie entre les lignes de vie abstraites et raffinées. Chaque ligne de vie abstraite doit être liée à une ou plusieurs lignes de vie qui peuvent être soit des lignes de vie non raffinées ou de nouvelles lignes de vie (des substituants ou des lignes de vie filles).

Definition 19 (Correspondance des lignes de vie)**.**
Nous définissons la fonction χ de correspondance des lignes de vie, qui est une fonction surjective partielle (notée $\twoheadrightarrow$) :

$$\chi : L_2 \twoheadrightarrow L_1$$

• Tous les messages abstraits doivent être présents dans le DS raffiné qui peut contenir éventuellement de nouveaux messages.

Definition 20 (Correspondance de messages)**.**
Nous définissons la fonction ρ, $\rho : M_2 \twoheadrightarrow M_1$, qui est une fonction bijective partielle telle que son domaine est l'ensemble des messages raffinés et le co-domaine est l'ensemble des messages abstraits.

• Pour la simplicité, nous surchargeons la fonction partielle ρ comme suit pour supporter les événements. Rappelons que EVT_i représente les événements du DSi, E_s_i représente les événements émis et E_r_i représente les événements reçus.

$$\rho : \left\{ \begin{array}{l} EVT_2 \rightarrowtail\!\!\!\!\!\rightarrow EVT_1 \\ \{(e,e')|e \in E\_s_2 \wedge e' \in E\_s_1 \wedge e' = FCT\_s_1(\rho(FCT\_s_2^{-1}(e)))\} \cup \\ \{(e,e')|e \in E\_r_2 \wedge e' \in E\_r_1 \wedge e' = FCT\_r_1(\rho(FCT\_r_2^{-1}(e)))\} \end{array} \right\}$$

• Nous introduisons les nouveaux ensembles New_M_2 et New_EVT_2 pour identifier respectivement les nouveaux messages et nouveaux événements qui sont introduits dans le diagramme de séquence raffiné (rappelons que $\rho^{-1}(M1)$ dénote les messages raffinés :

$$New\_M_2 = M_2 \setminus \rho^{-1}(M_1)$$
$$New\_EVT_2 = EVT_2 \setminus \rho^{-1}(EVT_1)$$

Proposition 1. *(Est-ce que $DS2 \sqsubseteq_\chi DS1$?)*
Soient $DS1$ et $DS2$ deux diagrammes de séquences dans DS. On définit deux fonctions χ et ρ. $DS2$ est un raffinement structurel de $DS1$ par rapport aux fonctions χ et ρ, $DS2 \sqsubseteq_{\chi,\rho} DS1$, si les conditions suivantes se produisent :

— *les messages abstraits dans $DS2$ sont reliés à leurs correspondants dans $DS1$ par le biais de la fonction*

$$\chi : M_1 \subseteq \rho(M_2)$$

— *les lignes de vie abstraites doivent garder leurs événements abstraits et les nouvelles lignes de vie héritent des événements abstraits de la ligne de vie substituée dans $DS2$, nous exprimons cette intuition formellement comme suit :*

$$FCT\_l_1 = \rho^{-1}; (EVT_1 \lhd FCT\_l_2); \chi$$

— *les nouveaux messages échangés entre les nouvelles lignes de vie dans DS2 sont reliés à la même ligne de vie abstraite, nous exprimons cette intuition formellement comme suit : :*

$$(New\_M_2) \lhd FCT\_s_2; FCT\_l_2; \chi = (New\_M_2) \lhd FCT\_r_2; FCT\_l_2; \chi$$

Chaque item de la prop.1 se réfère à une ou plusieurs règles définies ci-dessus. Le premier item se réfère à la règle $R6$, le second item se réfère aux règles $R2, R1$ et le troisième item se réfère à la règle $R3$.
Cette relation est conforme avec la relation de raffinement de la méthode B événementiel. En effet, la relation de raffinement que nous avons définie est basée sur la traduction d'alphabet par opposition à l'extension d'alphabet. La traduction d'alphabet [Boi14] nécessite une correspondance explicite entre les composants des diagrammes de séquence considérés.
Raffinement sémantique. La relation de raffinement est définie comme une simulation sur les sémantiques opérationnelles. Deux diagrammes peuvent être comparés s'ils possèdent le même alphabet. Par conséquent, les nouveaux événements dans le digramme de séquence raffiné sont considérés comme silencieux et ils sont remplacés par les τ transitions.

Definition 21 (Correspondance des états)**.**
Soient $Sem(DS1)$ et $Sem(DS2)$ les expressions des deux sémantiques opérationnelles associées à une paire de diagrammes de séquence. La relation de correspondance R entre une paire de sémantique opérationnelle consiste à lier leurs états respectifs S_i par rapport aux fonctions χ et ρ. S_i dépend des valeurs des deux variables $current\_lifeline\_i$ et $state\_i$; où $i \in \{1, 2\}$

— *Chaque ligne de vie raffinée doit être reliée à une ligne de vie abstraite.*

$$current\_lifeline\_1 = \chi(current\_lifeline\_2)$$

— *en faisant abstraction aux nouveaux événements introduits, chaque événement abstrait, dans le diagramme de séquence raffiné, doit être relié dans le diagramme de séquence abstrait :*

$$(New\_EVT_2) \lhd state\_2 = \rho; state$$

Proposition 2. *(Est ce que $DS2 \sqsubseteq_R DS1$?)*
Soient $DS1$ et $DS2$ deux diagrammes de séquence tels que $DS2$ est un raffinement structurel du diagramme de séquence $DS1$ ($DS2 \sqsubseteq_{\chi,\rho} DS1$), et $Sem(DS_i) = \langle S_i, s_i^0, \longrightarrow \rangle$ leurs sémantiques opérationnelles. De plus, les nouveaux événements de $DS2$ doivent terminer. $DS2$ est un raffinement de $DS1$ par rapport à la relation de correspondance $R(\rho, \chi)$ noté $DS2 \sqsubseteq_R DS1$ si les conditions suivantes se produisent :

— *les états initiaux sont liés : $(s_1^0, s_2^0) \in R$,*

— *pour chaque exécution d'un événement e', dans $DS2$, qui raffine un événement abstrait $e' \in \rho^{-1}(M_1)$, il existe une exécution de $\rho(e)$ dans $DS1$ qui lui est reliée, avec renforcement des gardes dans le cas où l'exécution d'un événement dépend de la garde :*

$$(\forall e \in \rho^{-1}(M_1))\ (\forall p \in S_1)\ \ (\forall q \in S_2)\ (\forall q' \in S_2)$$
$$[(q,p) \in R\ \wedge\ q \xrightarrow{[g]e}_2 q' \Rightarrow (\exists p' \in S_1)[p \xrightarrow{[g']\rho(e)}_1 p'\ \wedge\ g \Rightarrow g'\ \wedge\ (q',p') \in R]]$$

— *les nouveaux événements ne changent pas l'état du diagramme de séquence abstrait.*

$$(\forall e \in New\_EVT_2)\ (\forall p \in S_1)\ (\forall q \in S_2)\ (\forall q' \in S_2)$$
$$[(p,q) \in R\ \wedge\ q \xrightarrow{e}_2 q' \Rightarrow (q',p) \in R$$

Propriétés préservées par notre relation de raffinement.
• Comme conséquence de la Prop.2, nous pouvons affirmer que nous avons un raffinement de trace [Mic05] tout en omettant les nouveaux événements dans $DS2$ tels que les comportements requis sont préservés.
• La relation de raffinement permet l'extension des traces en faisant la distinction entre les événements observables et non-observables.
• Elle est réflexive, puisque nous avons $DS1 \sqsubseteq_R DS1$ qui est toujours vrai.
• Elle est transitive, puisque $DS_2 \sqsubseteq_R DS1$ et $DS3 \sqsubseteq_Q DS2$ implique $DS3 \sqsubseteq_{R;Q} DS1$.
• Finalement elle est substitutive car elle supporte le renommage des composants du DS abstrait dans le diagramme de séquence raffiné.

5.4.3 Problèmes induits par le raffinement

Il y a quelques problèmes qui sont induits par quelques types de raffinement.

— Le premier problème est causé par le raffinement vertical lorsque les événements abstraits qui appartiennent à une ligne de vie dans le DS abstrait sont distribués entre, au moins deux nouveaux lignes de vie qui substituent une ligne de vie abstraite. Une conséquence de cette distribution des événements est la perte de l'ordre causal entre eux, par conséquent nous obtenons plus de traces dans le DS concret dont quelques unes parmi eux n'ont pas leurs homologues dans DS1. Pour remédier à ce problème et rétablir l'ordre perdu, nous proposons une correction au diagramme de séquence DS2 qui consiste à ajouter le FC STRICT pour couvrir ces événements. Ce problème est bien connu dans la littérature il est nommé *race condition* [Chi05], les auteurs proposent d'ajouter des messages silencieux entre ces événements. Ce qui revient au même.

Illustration 1 : considérons le DS abstrait de la Figure 5.16, en raffinant la ligne de vie $L3$ qui est substituée par les lignes de vie $L3.1$ et $L3.2$, les événements qui appartiennent à la ligne de vie $L3$ ($!m1$ et $!m2$) sont distribués entre les lignes de vie $L3.1$ et $L3.2$ (Figure 5.17). Dans le DS abstrait, nous avons la contrainte de causalité $!m1 <!m2$,

toutes les traces possibles du $DS1$ doivent respecter cette contrainte. Dans le DS raffiné, la distribution de événements engendre la perte de cet ordre entre les événements $!m1$ et $!m2$ d'où on obtient plus de traces dans le DS raffiné qui n'ont pas leurs contreparties dans le DS abstrait.

— le même problème de la perte de relations de précédence apparaît lorsque nous avons un raffinement de messages par décomposition et qu'une réception d'un message fait perdre la relation de précédence $<_{EE}$ entre des messages abstraits.

Illustration 2 : considérons le DS abstrait de la Figure 5.18, les événements $!m1$ et $!m2$ sont reliés par la relation $<_{EE}$. Pour le message $m3$, puisque nous sommes dans un contexte de composants distribués et contrairement aux définitions de la sémantique standard qui ordonnent les événements le long de chaque ligne de vie, le message $m3$ peut être reçu soit avant, soit après soit entre les messages $m1$ et $m2$. Le message $m3$ est raffiné en deux messages $m3.1$ et $m3.2$, les Figures 5.19, 5.20, 5.21, 5.22, 5.23 représentent les positions possibles des réceptions des événements $m3.1$ et $m3.2$. Dans les diagrammes de séquence D3, D4 et D5, nous avons une perte de l'ordre entre les événements $!m1$ et $!m2$.

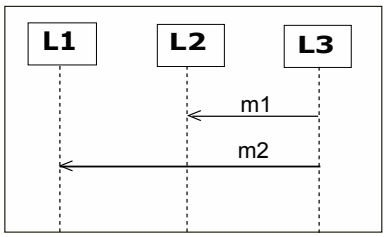

FIGURE 5.16 – DS0 abstrait

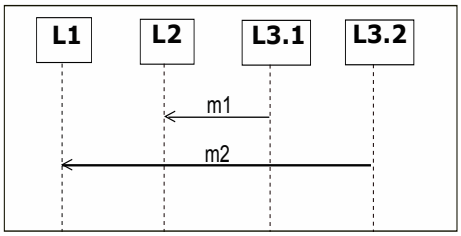

FIGURE 5.17 – DS1 : raffinement du DS0 de la Figure 5.16

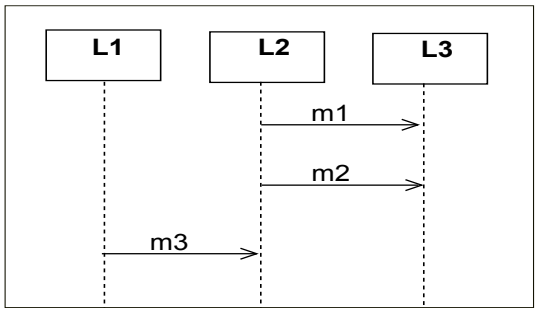

FIGURE 5.18 – D1 : Diagramme de séquence abstrait

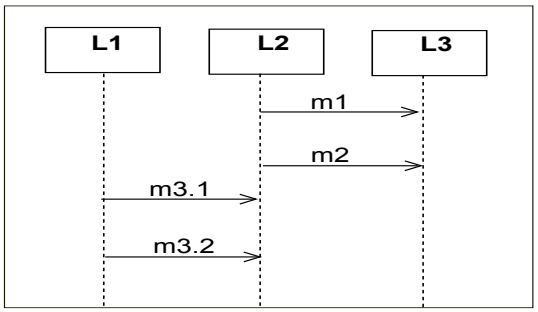

FIGURE 5.19 – D2 : Raffinement possible de DS de la Figure 5.18

— Le second problème est causé par le renforcement des gardes des opérandes.

Illustration 3 : considérons les DS représentés par les Figures 5.24 et 5.25. Dans le premier DS1 de la Figure 5.24, le message m peut se déclencher lorsque la condition $C > 0$. Dans le DS2 de la Figure 5.25, la condition est plus déterministe en restreignant son intervalle de validité entre 10 et 50. Nous avons aussi moins de traces en effet les traces correspondant vérifiant une condition hors l'intervalle $[10, 50]$ sont perdues dans le DS2.

Dans le Chapitre implémentation 7, nous proposons des solutions pour corriger ces problèmes.

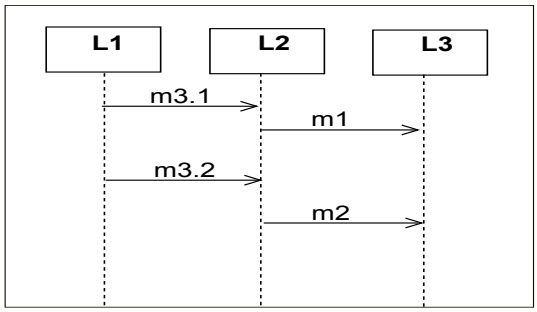

FIGURE 5.20 – D3 : Raffinement possible de DS de la Figure 5.18

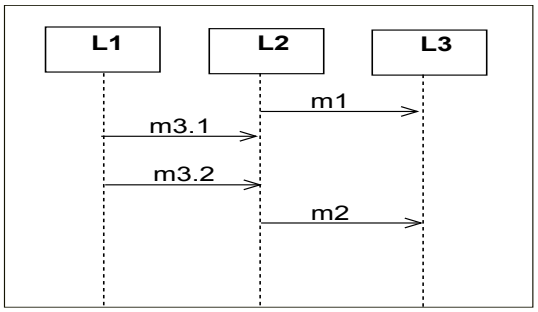

FIGURE 5.21 – D4 : Raffinement possible de DS de la Figure 5.18

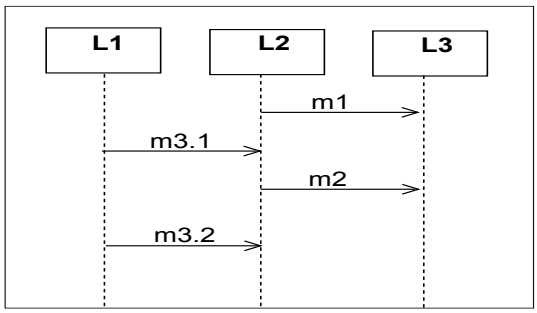

FIGURE 5.22 – D5 : Raffinement possible de DS de la Figure 5.18

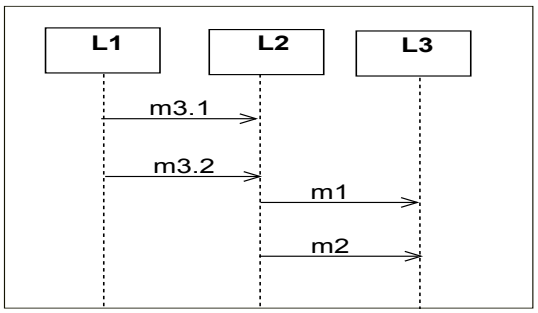

FIGURE 5.23 – D6 : Raffinement possible de DS de la Figure 5.18

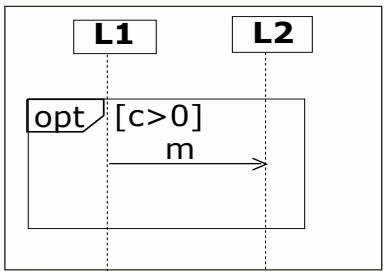

FIGURE 5.24 – DS abstrait

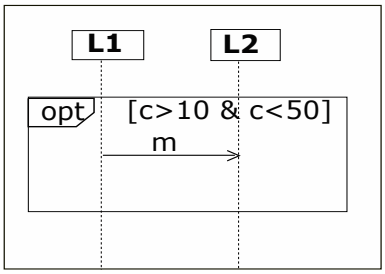

FIGURE 5.25 – DS raffiné : renforcement de la garde de l'opérande OP11

5.5 Conclusion

Dans ce chapitre nous avons traité le raffinement des diagrammes de séquence qui sont équipés avec la sémantique opérationnelle que nous avons déjà définie. Notre approche présente les avantages suivantes : nous avons considéré le raffinement structurel et sémantique :

i) en définissant clairement au préalable les règles qui régissent un raffinement structurel correct. Ce qui permet de guider le concepteur,
ii) en proposant une méthode cohérente et intuitive pour la formalisation de la relation de raffinement. Cette dernière possède les propriétés propices à un développement incrémental ainsi qu'à la réutilisation des diagrammes de séquence et leur instanciation,
iii) notre approche de formalisation de la relation de raffinement pour les diagrammes de séquence UML2.X n'est liée à aucun formalisme cible et peut être implémentée par n'importe quel outil.
Ce travail a fait l'objet d'une publication dans une conférence internationale [Fat16].

Chapitre 6

Implémentation de la sémantique opérationnelle en B événementiel

Sommaire

6.1 Introduction

Le couplage d'une méthode formelle et d'une méthode semi-formelle est une pratique de modélisation très répandue permettant de tirer profits des avantages des deux méthodes. Dans une étape préliminaire de modélisation, les travaux de l'état de l'art se basent soit sur des méthodes semi-formelles, en intégrant dans une étape suivante une méthode formelle, soit l'inverse.

Dans le cadre de nos travaux, nous utilisons les diagrammes de séquence d'UML2.X comme méthode semi-formelle, puis nous intégrons la méthode formelle B événementiel.
Il existe une multitude de travaux autour de la dérivation de UML vers B dont les objectifs peuvent se cerner comme suit : lever des ambiguïtés liées à la sémantique d'UML et/ou obtenir une modélisation rigoureuse et analysable par les outils de vérification dédiés par l'environnement de la méthode B. Ces travaux peuvent être catégorisés en deux approches :
• l'approche compilée qui est fondée sur la proposition des règles de traduction directes du modèle semi-formel vers des spécifications formelles. Notons que la plupart des travaux ([Eri99], [F. 06a]) se basent sur cette approche. Cependant, quelques uns ont abouti à la proposition des outils de traduction : UML2SQL [Siw10], U2B [F. 03], Argo UML+B [H.02],
• l'approche interprétée qui se base sur la proposition d'une formalisation du méta-modèle du modèle semi-formel vers des spécifications formelles ([LP02]).

Dans le cadre de nos travaux, que nous appliquons l'approche compilée en proposant des règles de traduction des diagrammes de séquence vers des spécifications B-événementiel.
Nous soulignons que dans la littérature les diagrammes d'UML qui ont été traités en suivant cette approche sont : les diagrammes de classes, les diagrammes états transitions et les diagrammes de collaborations.

Justification du choix de la méthode formelle B-événementiel. B événementiel est dotée d'une sémantique bien définie [J.-96b] et de plusieurs aspects clés qui justifient notre choix pour ce formalisme. Le premier aspect est l'utilisation de la théorie des ensembles comme notation de modélisation ; ceci permet une traduction intuitive de la sémantique des DS dont la formalisation des règles est également basée sur la théorie des ensembles ainsi que sur la logique des prédicats. Le second aspect réside dans le fait que B événementiel est un système qui est basé sur les événements notion prouvée utile dans l'analyse des besoins dans la modélisation de systèmes distribués, dans la conception des algorithmes et dans la programmation séquentielle et distribuée. Dans le modèle B événementiel, le choix de l'exécution des événements est non déterministe, il suit le principe de entrelacement (*interleaving*) tout en présumant l'atomicité des événements.

Ces critères répondent aux exigences de la sémantique causale. Puisque c'est une sémantique de entrelacement (*interleaving semantics*) et nous avons besoin d'une exécution non déterministe des événements. Ceci est particulièrement utile, pour capter les spécificités de la sémantique standard de certains FC tels que le fragment combiné ALT. Ces exigences ne sont pas captées explicitement dans d'autres formalismes.

Avec le B événementiel, nous pouvons exprimer et vérifier des propriétés importantes telles que des propriétés de sûreté, de vivacité et des modalités [L.M98]. Dans la littérature nous retrouvons plusieurs travaux qui se sont intéressés à l'étude des propriétés des systèmes en utilisant le B événementiel [BB06], [BB02], [Ol07], [Fam13], [Fam14], [Ste14]. D'autres propriétés ont été exploitées, ce sont les propriétés d'atteignabilité et d'équité en associant soit la logique modale (LTL ou CTL) soit le langage de spécification TLA+ qui est un langage de spécification qui étend la logique temporelle des actions TLA à l'aide de la structure de module et la théorie des ensembles [Lam02].

6.2 Traduction des diagrammes de séquence en spécifications B événementiel : modèle générique de traduction

Nous traduisons les diagrammes de séquence d'UML2.X munis de leur sémantique opérationnelle que nous avons définie dans le Chapitre 4 vers des spécifications en B événementiel.

Afin d'aboutir à une architecture générique de traduction des diagrammes de séquence vers des spécifications en B événementiel, nous avons procédé par étapes en considérant des diagrammes de séquence avec des caractéristiques différentes. En effet, nous avons considéré d'abord des diagrammes de séquence basiques, ensuite des diagrammes de séquence avec des fragments combiné SEQ, ALT et OPT, LOOP, STRICT et PAR en expérimentant les différentes combinaisons ainsi que les dispositions possibles des FC considérés (séquentiels, imbriqués, imbriqués et séquentiels).
La définition formelle d'un DS est exprimée dans les contextes et le comportement du DS que nous avons défini dans la sémantique opérationnelle est captée dans la machine B. Pour n'importe quel diagramme de séquence, nous définissons trois contextes et une machine B. La Figure 6.1 représente l'architecture générique que nous proposons pour la traduction de n'importe quel DS vers des spécifications en B événementiel. Cette structuration permet d'alléger la spécification et de faciliter les preuves de chaque composant du modèle B. Dans ce qui suit, nous détaillons le contenu de chaque composant.

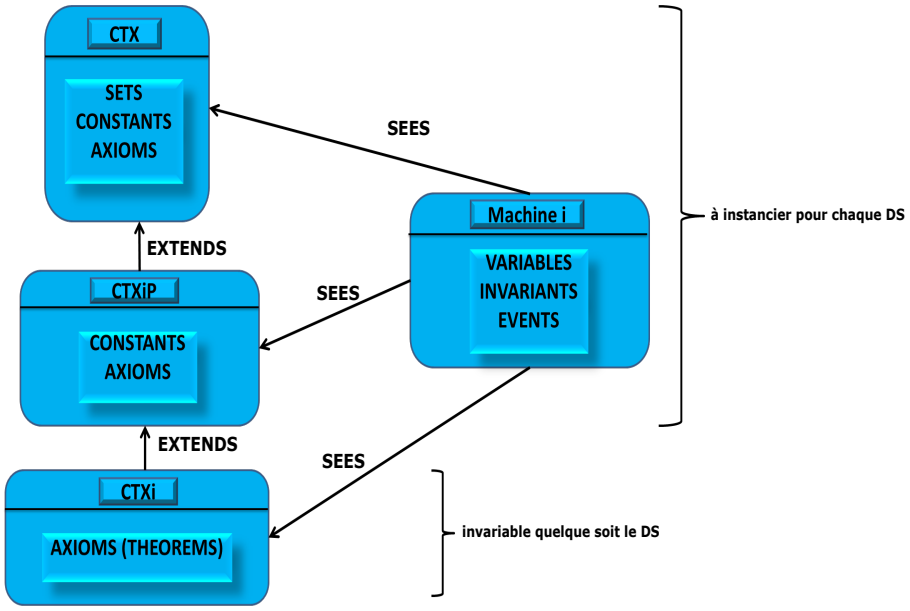

FIGURE 6.1 – L'architecture générique de la traduction

6.2.1 Construction des contextes

Considérons un diagramme de séquence avec les composants suivants :

$$\langle L, M, EVT, FCT\_s, FCT\_r, FCT\_l, OP, F, <_{caus}, tree\_OP \rangle$$

N.B. Pour les événements nous faisons la distinction entre les événements reçus et émis respectivement dans les éléments E_r et E_s.
Le diagramme de séquence considéré est équipé avec sa sémantique opérationnelle

$$Sem(DS) = \langle S, S^0, \longrightarrow \rangle$$

Nous explicitons dans le Tableau 6.1 la correspondance entre chaque composant du diagramme de séquence et son codage en B. Dans le Tableau 6.2 nous donnons correspondance entre chaque relation de précédence de la sémantique causale et son codage en B.

Nous traduisons la définition formelle d'un DS dans des contextes via des constantes, des ensembles et des axiomes. Dans les contextes, nous séparons les caractéristiques communes à tous les diagrammes de séquence qui peuvent être exprimées de façon générale et celles qui sont propres à chaque diagramme de séquence. Ainsi, dans le premier et le second contexte (CTX, $CTXiP$[8]), nous exprimons les valeurs concrètes des composants du DS considéré (lignes de vie, messages, événements, opérandes) qui sont définis par des constantes et des ensembles. Le contexte $CTXiP$ étend (EXTENDS) le contexte CTX. Dans le troisième contexte ($CTXi$), qui étend le deuxième contexte, nous définissons les axiomes ; ils permettent de spécifier le typage des constantes ou d'y exprimer des propriétés. Par conséquent, ils sont marqués comme des théorèmes ; leurs preuves nous permet de vérifier que le DS considéré est bien formé. Le contenu de ce contexte est générique, nous détaillons son contenu ci-dessous.

Considérons le contexte $CTXi$ représenté par la Figure 6.2 :
• nous exprimons dans les axiomes 1..14 que les ensembles représentant les lignes de vie, les messages, les événements ainsi que les opérandes sont non vides et finis. Nous exprimons aussi que tous les messages d'un diagramme de séquence sont distincts.

$$\forall (m, m') \cdot (m, m') \in X\_MSG \Longrightarrow m \neq m'$$

8. i est le numéro de la spécification

Élément du DS	Code B	Élément du DS	Code B
L	$LIFELINES$	M	MSG
E_s	EVT_S	E_r	EVT_R
EVT	EVT	FCT_s	FCT_EVT_S
FCT_r	FCT_EVT_R	FCT_l	FCT_l
OP	$OPERANDS$	F	
$<_{Sync}$	$causync$	$<_{RE}$	$causRE$
$<_{EE}$	$causEE$	$<_{SyncG}$	$causyncG$
$<_{causG}$	$causG$	$<_{REG}$	$causREG$
$<_{EEG}$	$causEEG$	$<_{Hcaus}$	$Hcaus$
$<_{HcausG}$	$HcausG$	$<_{\tau 1}$	$causT1$
$<_{\tau 2}$	$causT2$	$<_{\tau 3}$	$causT3$
$<_{caus}$	$caus$		

Table 6.1 – Correspondance entre chaque élément du DS et son codage en B événementiel

Relation de précédence	Code B	Relation de précédence	Code B
$<_{Sync}$	$causync$	$<_{RE}$	$causRE$
$<_{EE}$	$causEE$	$<_{SyncG}$	$causyncG$
$<_{causG}$	$causG$	$<_{REG}$	$causREG$
$<_{EEG}$	$causEEG$	$<_{Hcaus}$	$Hcaus$
$<_{HcausG}$	$HcausG$	$<_{\tau 1}$	$causT1$
$<_{\tau 2}$	$causT2$	$<_{\tau 3}$	$causT3$

Table 6.2 – Correspondance entre chaque relation de précédence de la sémantique causale et son codage en B événementiel

Cette propriété est obtenue systématiquement puisque X_MSG[9] est une partition de l'ensemble $MESSAGES$ donc tous ses éléments sont distincts.

• nous exprimons dans l'axiome 23 que chaque événement est affecté à une seule ligne de vie,

• nous exprimons dans les axiomes 17 et 18 que chaque événement est associé à son message correspondant,

• nous exprimons dans les axiomes 22 et 23 des propriétés sur la fonction FCT_l qui permet de spécifier pour chaque événement sa ligne de vie émettrice ou réceptrice, qui est une fonction surjective et non vide,

• dans l'axiome 16 et 17, nous exprimons respectivement le typage des fonctions bijectives FCT_s et FCT_r ; ces fonctions permettent respectivement d'associer à chaque message son événement d'envoi et de réception,

• nous définissons la relation de causalité X_caus dans laquelle nous explicitons les relations de précédence de chaque événement avec les autres événements. Cette relation est nécessairement non-réflexive. Le typage de la relation de causalité est donné dans l'axiome 18. L'axiome 20 exprime que la relation d'ordre causale est non réflexive et acyclique,

• nous définissons la constante X_Begin dans laquelle nous précisons l'événement de début, son typage est exprimé dans l'axiome 20. L'événement de début possède la propriété de ne pas avoir de précédents, nous exprimons ceci avec l'axiome 19,

• nous définissons dans l'axiome 24 la fonction X_WEIGHT qui permet d'associer à chaque événement son poids, qui correspond à son nombre maximum d'occurrences pour chaque exécu-

9. Le préfixe X peut être instanciée par A pour désigner qu'il s'agit de la spécification abstraite, Rn pour désigner qu'il s'agit du n^{ime} raffinement de la spécification

tion.

CONTEXT CTXi
EXTENDS CTXiP
AXIOMS
AXIOMS

axm1: $X\_LIFELINES \subseteq LIFELINES$
axm2: $X\_LIFELINES \neq \varnothing$
axm3: $card(X\_LIFELINES) \geq 1$
axm4: $X\_MSG \subseteq MSG$
axm5: $X\_MSG \neq \varnothing$
axm6: $card(X\_MSG) \geq 1$
axm7: $X\_EVT \subseteq EVT$
axm8: $X\_EVT \neq \varnothing$
axm9: $X\_OPERANDS \subseteq OPERANDS$
axm10: $card(X\_OPERANDS) \geq 1$
axm11: $X\_OPERANDS \neq \varnothing$
axm12: $X\_EVTG \subseteq EVT$
axm13: $X\_TEVT \subseteq EVT$
axm14: $X\_EVT\_S \cap X\_EVT\_R = \varnothing$
axm15: $X\_EVT = X\_EVT\_s \cup X\_EVT\_r$
axm16: $FCT\_s \in X\_MSG \rightarrowtail\!\!\!\!\!\!\twoheadrightarrow X\_EVT\_s$
axm17: $FCT\_r \in X\_MSG \rightarrowtail\!\!\!\!\!\!\twoheadrightarrow X\_EVT\_r$
axm18: $X\_caus \in X\_EVT \leftrightarrow X\_EVT$
axm19: $X\_caus \cap (X\_EVT \lhd id) = \varnothing$
axm20: $X\_Begin \subset X\_EVT$
axm21: $X\_EVT \setminus ran(caus) = X\_Begin$
axm22: $FCT\_l \in X\_EVT \twoheadrightarrow X\_LIFELINES$
axm23: $FCT\_l \neq \varnothing$
axm24: $X\_WEIGHT \in X\_EVTG \rightarrow \mathbb{N}$

END

FIGURE 6.2 – Théorèmes du contexte $CTXi$: partie générique du codage

6.2.2 Construction de la machine B événementiel

La sémantique opérationnelle du DS, $Sem(DS)$, est traduite dans la machine B telle que : i) l'état S du DS est exprimé avec deux variables : la variable *state*, qui est une fonction bijective totale qui exprime l'état de chaque événement et la variable $current\_lifeline$ qui exprime la ligne de vie émettrice ou réceptrice de l'événement en cours ; ii) chaque transition qui appartient à $\longrightarrow$ est représentée par un événement dans la machine B.

La machine B possède une visibilité sur les déclarations statiques qui se trouvent dans les contextes $CTX, CTXi$ et $CTXiP$ par les relations SEES. La partie statique de la machine encapsule : *i)* les variables d'états qui sont génériques pour tous les DS ($state$ et $current\_lifeline$), d'autres variables spécifiques au DS encodé peuvent être définies ; *ii)* le typage des variables est

fourni dans l'invariant de la machine.

La partie dynamique de la machine est composée par l'initialisation des variables et les événements. Les événements de la machine sont les événements du DS. Pour un DS avec FC gardés nous rajoutons les événements fictifs qui sont essentiellement requis pour l'évaluation des gardes et la synchronisation entre les lignes de vie.

Pour chaque événement d'un DS d'UML2.X, les conditions de déclenchement CD1 et CD1' [10] que nous avons définies dans le Chapitre sémantique opérationnelle 8.2, sont exprimées avec plusieurs gardes qui doivent être vérifiées conjointement. Le nombre de gardes dépend des différentes localisations des événements précédents de l'événement en cours, puisqu'ils sont regroupés par opérande. La troisième condition de déclenchement dans laquelle nous vérifions que l'événement est encore déclenchable, est exprimée avec une seule garde. La quatrième condition de déclenchement dans laquelle nous vérifions la valeur de la contrainte dinteraction (garde) de l'opérande est exprimée avec une garde.
Concernant les actions de chaque événement : chaque action de chaque événement est un prédicat qui est composé d'un ou plusieurs effet dexécution EEi que nous avons définis dans le Chapitre sémantique opérationnelle 8.2. Nous récapitulons les actions de chaque type d'événement d'un diagramme de séquence d'UML2.X dans le Tableau 6.3.

	Action1	Action2
Événement normal	EE1	EE2
Événement fictif positif d'un opérande d'un FC ALT	EE1, EE1', EE3	EE2
Événement fictif négatif d'un opérande d'un FC ALT	EE1, EE3'	EE2
Événement fictif positif d'un opérande d'un FC LOOP	EE1, EE1'	EE2
Événement fictif négatif d'un opérande d'un FC LOOP	EE1, EE3"	EE2

TABLE 6.3 – Les actions de chaque type d'événement

Pour récapituler, le contenu du contexte $CTXi$ est invariable pour n'importe quel DS. Dans les contextes CTX et $CTXiP$, les valeurs des ensembles et des constantes doivent être instanciées avec les valeurs du DS considéré. Dans la machine, quelques variables (spécifiquement ceux qui expriment l'état du DS) ainsi que leurs invariants sont invariables. Désormais l'implémentation d'un DS avec sa sémantique opérationnelle, qui est basée sur la sémantique causale, vers des spécifications en B événementiel peut être utilisée comme un support pour le raisonnement pour des exemples variés de spécifications des DS.

6.3 Expérimentations

Pour illustrer l'utilisation de la sémantique causale et la sémantique opérationnelle, nous considérons des DS avec différentes spécificités comme des spécifications d'entrée (*input*) pour l'analyse.

6.3.1 Traduction d'un diagramme de séquence basique

Considérons le diagramme de séquence, représenté par la Figure 6.3, qui est constitué par trois lignes de vie indépendantes échangeant des interactions basiques (sans aucun FC). Nous représentons le contenu de chaque contexte (CTX et $CTX0P$) respectivement dans les Figures 6.4

10. Rappelons que CD1 et CD1' sont les conditions de déclenchement dans lesquelles nous vérifions l'occurrence des événements précédents de l'événement en cous.

et 6.5. Le squelette de la machine de la spécification relative au DS considéré est représenté dans la Figure 6.6.
Les événements de la machine $M0$ sont tous implémentés de façon similaire puisqu'ils sont simples. Les conditions de déclenchement qui sont codées par des gardes dans la machine B consistent à vérifier que les événements précédents de l'événement courant se sont déclenchés ($grd1$) et que l'événement ne s'est pas encore déclenché ($grd2$). Les effets d'exécution de l'événement qui sont représentés par des actions consistent respectivement à mettre à jour l'état de l'événement en décrémentant sa variable d'état *state* ($act1$) et à mettre à jour sa ligne de vie dans la variable $current\_lifeline$ ($act2$). Nous donnons l'implémentation de l'événement e_m2 dans la Figure 6.7.

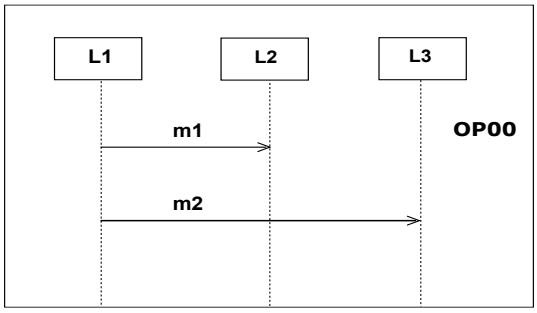

FIGURE 6.3 – Diagramme de séquence basique

CONTEXT CTX
SETS
INSTANCES , MSG, EVT, OPERANDS
CONSTANTS
L1 , L2, L3, m1, m2, e_m1, e_m2, r_m1, r_m2
AXIOMS
axm1: $partition(LIFELINES,\ \{L1\},\ \{L2\},\ \{L3\})$
axm2: $partition(MSG, \{m1\}, \{m2\})$
axm3: $partition(EVT, \{e\_m1\}, \{e\_m2\}, \{r\_m1\}, \{r\_m2\})$
axm4: $partition(OPERANDS, \{OP00\})$
END

FIGURE 6.4 – Context CTX : codage de la partie statique du diagramme de séquence de la Figure 6.3

6.3.2 Traduction d'un diagramme de séquence avec FC imbriqués

Nous reprenons le diagramme de séquence (voir Figure 3.24) relatif au système de gestion de base de données que nous avons présenté dans le Chapitre 3 comme entrée (*input*) pour l'analyse.

L'architecture de l'étude de cas obtenu à partir de ProB est illustrée dans la Figure 6.8. Les Figures 6.9, 6.10, 6.11 représentent respectivement l'implémentation des contextes CTX, $CTX1P$ et le squelette de la machine $M1$ du modèle générique que nous avons proposé. Dans

CONTEXT CTX0P
EXTENDS CTX
AXIOMS
AXIOMS
axm1: $causSync = \{(e\_m1 \mapsto r\_m1), (e\_m2 \mapsto r\_m2)$
axm2: $causEE = \{(e\_m1 \mapsto e\_m2)\}$
axm3: $causRE = \varnothing$
axm4: $caus = causSync \cup causEE \cup causRE$
axm5: $Begin = \{e\_m1\}$
axm6: $A\_OPERANDS = \{OP00\}$
axm7: $A\_WEIGHT = \{(e\_m1 \mapsto 1), (e\_m2 \mapsto 1), (r\_m1 \mapsto 1), (e\_m2 \mapsto 1), (r\_m1 \mapsto 1), (r\_m2 \mapsto 1)\}$
axm8: $EVT\_OP00 = \{e\_m1, e\_m2, r\_m1, r\_m2\}$
END

FIGURE 6.5 – le contexte $CTX0P$ avec quelques axiomes de la spécification

le contexte $CTX1P$ (Figure 6.10), les axiomes $axm1..axm12$ expriment le calcul des relations de précédence $<_{Sync}$, $<_{RE}$, $<_{EE}$ et $<_{Hcaus}$ que nous avons explicitées dans le Chapitre 3. Dans l'axiome 21 de la Figure 6.10, nous associons à chaque événement son nombre maximum d'occurrence dans un *run* (son poids). Dans les axiomes $axm22..axm26$ de la Figure 6.10, nous associons à chaque opérande ses événements.

Dans la machine, nous définissons les variables *state*, $current\_lifeline$ et gi, qui expriment respectivement l'état de chaque événement, la ligne de vie émettrice ou réceptrice de l'événement courant et les gardes des opérandes. Le typage des variables est défini dans la clause INVARIANT. Dans la clause INITIALISATION, nous initialisons l'état de chaque événement avec son poids qui correspond à son nombre maximal d'occurrence dans chaque exécution (*run*). La variable $current\_lifeline$ est initialisée avec la ligne de vie de l'événement qui est autorisé à se déclencher et qui est défini dans la variable $Begin$ dans l'axiome 19 de la Figure 6.10.

Nous représentons le codage de l'implémentation des événements les plus significatifs de l'étude de cas : un événement normal (voir Figure 6.15), un événement fictif positif et négatif d'un fragment combiné ALT (respectivement dans les Figures 6.14 et 6.16) et d'un fragment combiné LOOP (respectivement dans les Figures 6.12 et 6.13). Ceci peut guider l'implémentation des événements de n'importe quel DS avec FC imbriqués. La Figure 6.15 représente l'implémentation d'un événement normal *r_connect* ayant des gardes et des actions très simples. Pour n'importe quel événement, nous vérifions dans les gardes, que l'événement se déclenche avec des conditions correctes.

En effet dans la première garde nous vérifions que ses événements précédents, qui sont localisés dans l'opérande $OP00$ se sont déclenchés ; dans la seconde garde nous vérifions que l'événement peut encore se déclencher. Après son occurrence nous mettons à jour son état ainsi que l'état de la variable $current\_lifeline$ respectivement dans la première et dans la deuxième action. La Figure 6.12 représente l'implémentation de l'événement fictif positif de l'opérande LOOP $OP21$ (T_OP21p). L'événement T_OP21p possède des relations de précédence qui sont localisées respectivement dans les opérandes $OP00$ et $OP11$ (axiomes 8, 9 dans la Figure 6.10) et des relations de précédence cachées dans les opérandes $OP21$, $OP31$ et $OP32$ (axiome 10 dans la Figure 6.10). Rappelons que les événements d'un opérande LOOP ou qui appartiennent à un FC

MACHINE M1
SEES CTX1
VARIABLES
state , current_lifeline
INVARIANTS
inv1: $state \in X\_EVTG \rightarrow \mathbb{N}$
inv2: $current\_lifeline \in X\_LIFELINES$
EVENTS
Initialisation
begin
act1: $state := X\_WEIGHT$
act2: $current\_instance := L1$
end
Event e_m1 ⟨ordinary⟩ $\widehat{=}$
when
grd1: ...
then
act1: ...
act2: ...
end
Event r_m1 ⟨ordinary⟩ $\widehat{=}$
when
grd1: ...
then
act1: ...
act2: ...
end
Event e_m2 ⟨ordinary⟩ $\widehat{=}$
when
grd1: ...
then
act1: ...
act2: ...
end
Event r_m2 ⟨ordinary⟩ $\widehat{=}$
when
grd1: ...
then
act1: ...
act2: ...
end
END

FIGURE 6.6 – Squelette de la machine B événementiel de l'étude de cas

Event e_m2 ⟨**ordinary**⟩ $\widehat{=}$

when

grd1: $\forall EE \cdot ((EE \in EVT\_OP00 \ \wedge EE \in ((caus)^{-1}[\{e\_m2\}] \Rightarrow ((state(EE) < state(e\_m2)))$

grd2: $state(e\_m2) \geq 1$

then

act1: $state(e\_m2) := state(e\_m2) - 1$

act2: $current\_lifeline := FCT\_l(e\_m2)$

end

FIGURE 6.7 – L'implémentation de l'événement e_m2

imbriqué contenant un opérande LOOP peuvent avoir, dans la première itération des relations de précédence (qui sont calculées à partir des relations $causSync$, $causEE$ et $causRE$), et à partir de la seconde itération ils peuvent avoir en plus des relations de précédence (qui sont calculées à partir des relations $causSync$, $causEE$ et $causRE$)), des relations de précédence cachées qui sont calculées à partir de la relation de causalité H_causG. Dans les trois premières gardes, nous vérifions que les événements précédents de l'événement T_OP21p se sont déclenchés ; par conséquent dans la première et la deuxième garde, nous vérifions que ses événements précédents (qui sont calculés à partir de la relation $(causG \setminus HcausG^{-1})$, dans la première itération, se sont déclenchés. Ensuite, dans la troisième garde, nous vérifions que ses événements précédents ainsi que ses événements précédents cachés (dont ces derniers sont calculés à partir de la relation $HcausG^{-1}$) à partir de la seconde itération se sont déclenchés. Dans la quatrième garde, nous vérifions que l'événement peut encore se déclencher, i.e il n'a pas encore atteint son nombre maximal d'occurrence. Dans la cinquième garde, nous vérifions que la garde de l'opérande considéré est évalué à vraie. Après son occurrence, nous mettons à jour son état (en décrémentant la variable $state$), ainsi que l'état de l'événement fictif négatif du même opérande ; nous mettons à jour l'état de la variable $current\_lifeline$ et nous initialisons la garde de l'opérande considéré $g2$.

Nous représentons l'implémentation de l'événement fictif négatif de l'opérande LOOP $OP21$ (T_OP21n) dans la Figure 6.13. L'événement T_OP21n possède quelques gardes similaires à celles de l'événement fictif positif $TOP21p$ (ce sont les trois premières gardes, dans lesquelles nous vérifions l'occurrence des événements précédents et la quatrième garde). Dans la cinquième garde nous vérifions que la garde de l'opérande $OP21$ est évaluée à fausse. Après son occurrence, dans la première action, nous décrémentons aussi bien son état ainsi que l'état de chaque événement de l'opérande considéré $OP31$, la deuxième et la troisième action sont similaires à ceux de l'événement fictif positif T_OP21p.
La Figure 6.14 représente l'implémentation de l'événement fictif positif de l'opérande ALT $OP31$ (T_OP31p). L'événement T_OP31p possède des relations de précédence dans l'opérande $OP21$. Dans la première garde nous vérifions que ses événements précédents qui sont localisés dans cette opérande se sont déclenchés. Dans la seconde garde, nous vérifions que l'événement peut encore se déclencher. Dans la troisième garde, nous vérifions que la garde de l'opérande considéré est évaluée à vraie. Après son occurrence, dans la première action, nous mettons à jour respectivement son état, l'état de l'événement fictif négatif du même opérande ainsi que les états respectifs des événements fictifs de l'opérande frère ($brother$) $OP32$. Dans la seconde action nous mettons à jour la variable $current\_lifeline$ et dans la troisième action nous initialisons la variable ($g3$) qui garde l'opérande considéré.

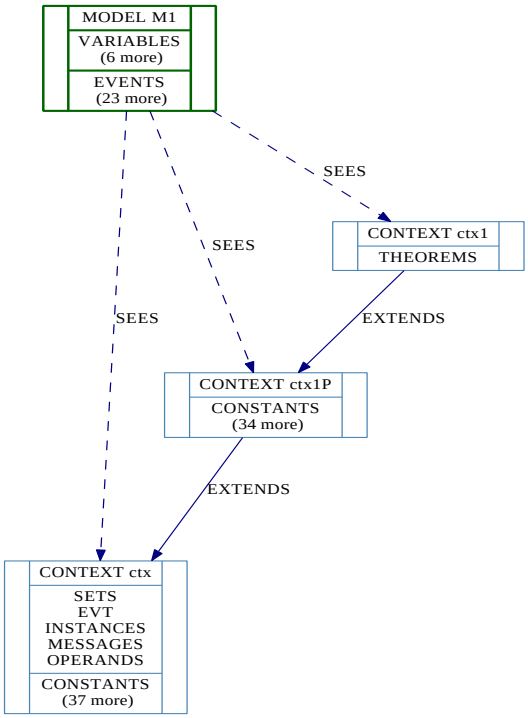

FIGURE 6.8 – L'architecture de l'étude de cas fournie ProB

La Figure 6.16 représente l'implémentation de l'événement fictif négatif de l'opérande ALT $OP31$ (T_OP31n). L'événement T_OP31p possède une première et une seconde garde qui sont similaires à celles de l'événement fictif positif T_OP31p. Dans la troisième garde, nous vérifions que la garde ($g3$) de l'opérande $OP31$ est évaluée à fausse. Après son occurrence, dans la première action, nous décrémentons aussi bien son état et l'état de chaque événement de l'opérande considéré $OP31$ pour empêcher leurs occurrences. La seconde et la troisième action sont similaires à celles de l'événement fictif positif T_OP31p.

6.3.3 Analyse formelle de la spécification

Pour vérifier, valider et analyser notre spécification, nous utilisons les outils offerts par la plateforme Rodin.

6.3.3.1 Vérification de cohérence avec Rodin

La correction d'un modèle B événementiel correct dépend essentiellement des preuves. Comme nous l'avons mentionné dans la Section A.2.3, une machine B possède plusieurs obligations de preuves générées automatiquement par Rodin. Les obligations de preuve sont uniquement générées à partir d'un modèle qui est statiquement cohérent (*sound*).

Dans notre cas les OPs non triviales ont été démontrées avec le mode automatique ou interactif. Dans le second mode, le concepteur est guidé par les démonstrateurs qui sont entièrement automatiques. Par conséquent, nous pouvons conclure que 100% des preuves ont été démontrées automatiquement.

La Table 6.4 illustre un récapitulatif des obligations de preuves relatives à l'étude de cas qui sont générées automatiquement par Rodin. Nous avons 169 obligations de preuves. Nous

CONTEXT **CTX**
SETS
LIFELINES , MSG, EVT, OPERANDS
CONSTANTS
$CLIENT, SERVER, DB, connect, input\_query, seek\_query, result\_query,$
$transmit\_result, not\_found, display\_not\_found, e\_connect, e\_input\_query,$
$e\_seek\_query, e\_result\_query, e\_transmit\_result, e\_not\_found,$
$e\_display\_not\_found, r\_connect, r\_input\_query, r\_seek\_query,$
$r\_result\_query, r\_transmit\_result, r\_not\_found, r\_display\_not\_found,$
$OP11, OP21, OP31, OP32, OP00, T\_OP11p, T11n,$
$T\_OP21p, T\_OP21n, T\_OP31p, T\_OP31n, T\_OP32p, T\_OP32n$
AXIOMS
axm1: $partition(LIFELINES, \{CLIENT\}, \{SERVER\}, \{DB\})$
axm2: $partition(MSG, \{connect\}, \{input\_query\}, \{seek\_query\},$
$\{result\_query\}, \{transmit\_result\}, \{not\_found\}, \{display\_not\_found\})$
axm3: $partition(EVT, \{e\_input\_query\}, \{e\_connect\}, \{e\_seek\_query\},$
$\{e\_result\_query\}, \{e\_transmit\_result\}, \{e\_not\_found\}, \{e\_display\_not\_found\},$
$\{r\_connect\}, \{r\_input\_query\}, \{r\_seek\_query\}, \{r\_result\_query\},$
$\{r\_transmit\_result\}, \{r\_not\_found\}, \{r\_display\_not\_found\}, \{T\_OP11p\},$
$\{T\_OP11n\}, \{T\_OP21p\}, \{T\_OP21n\}, \{T\_OP31p\},$
$\{T\_OP31n\}, \{T\_OP32p\}, \{T\_OP32n\})$
axm4: $partition(OPERANDS, \{OP00\}, \{OP11\}, \{OP21\}, \{OP31\}, \{OP32\})$
END

FIGURE 6.9 – Contexte CTX : Codage de la partie statique de l'étude de cas

distinguons quatre types d'obligations de preuves *THM, WD, FIS* et *INV*. Elles représentent respectivement les OP des théorèmes, de la bonne définition (well-definedness), de la faisabilité et de la préservation d'invariant.

Composants	THM	WD	FIS	INV	Pr %
CTX	0	0	*	*	100
CTX1	15	1	*	*	100
CTX1P	0	0	*	*	100
M1	0	94	11	48	100

TABLE 6.4 – Statistiques des obligations de preuves de l'étude de cas

6.3.3.2 Validation de la spécification avec ProB

ProB permet une analyse qui est basée sur l'exploration sur l'espace des états à partir d'un état initial plutôt que sur les preuves directes (*direct proof*). Par conséquent nous faisons appel à ProB en premier lieu pour une vérification rapide d'une spécification. La Figure 6.17 représente l'interface ProB qui est relative à l'étude de cas.

En appliquant l'explorateur de modèles ProB sur la machine de la spécification du DS encodé, ProB génère UN diagramme état transition qui contient l'ensemble des états ainsi que les transitions possibles, la Figure 6.25 illustre quelques statistiques sur le diagramme état transition que nous récapitulons dans le Tableau 6.5. Chaque état comprend tous les événements

11. états qui sont atteints à partir des nœuds qui atteints à partir des nœuds ouverts par les opérations permises.
12. États déjà calculés par ProB

CONTEXT **CTX1P**
EXTENDS **CTX**
AXIOMS

axm1: $causSync = \{(e\_connect \mapsto r\_connect), (e\_input\_query \mapsto r\_input\_query), (e\_seek\_query \mapsto r\_seek\_query), (e\_result\_query \mapsto r\_result\_query), (e\_transmit\_result \mapsto r\_transmit\_result), (e\_not\_found \mapsto r\_not\_found), (e\_display\_not\_found, r\_display\_not\_found\}$

axm2: $causEE = \{(e\_connect \mapsto e\_input\_query)\}$

axm3: $causRE = \{(r\_input\_query \mapsto e\_seek\_query), (r\_seek\_query \mapsto e\_result\_query), (r\_seek\_query \mapsto e\_not\_found), (r\_result\_query \mapsto e\_transmit\_result), (r\_not\_found \mapsto e\_display\_not\_found), (r\_connect \mapsto e\_seek\_query)\}$

axm4: $HcausOP11 = \{(e\_input\_query \mapsto e\_input\_query)\}$

axm5: $HcausOP21 = \{(e\_display\_not\_found \mapsto e\_seek\_query), (e\_transmit\_result \mapsto e\_seek\_query), (e\_seek\_query \mapsto e\_seek\_query)\}$

axm6: $Hcaus = HcausOP11 \cup HcausOP21$

axm8: $causT1EE = \{(e\_connect \mapsto T\_OP11p), (e\_connect \mapsto T\_OP11n)\}$

axm9: $causT1RE = \{(r\_connect \mapsto T\_OP21p), (r\_connect \mapsto T\_OP21n), (r\_input\_query \mapsto T\_OP21p), (r\_input\_query \mapsto T\_OP21n), (r\_seek\_query \mapsto T\_OP31p), (r\_seek\_query \mapsto T\_OP31n), (r\_seek\_query \mapsto T\_OP32p), (r\_seek\_query \mapsto T\_OP32n)\}$

axm10: $HcausT1 = \{(e\_display\_not\_found \mapsto T\_OP21p), (e\_display\_not\_found \mapsto T\_OP21n), (e\_transmit\_result \mapsto T\_OP21p), (e\_transmit\_result \mapsto T\_OP21n), (e\_input\_query \mapsto T\_OP21p), (e\_input\_query \mapsto T\_OP21n), (e\_seek\_query \mapsto T\_OP21p), (e\_seek\_query \mapsto T\_OP21n)\}$

axm11: $causT1 = \{causT1EE \cup causT1RE \cup HcausT1\}$

axm12: $causT2 = \{(T\_OP11p \mapsto e\_input\_query), (T\_OP21p \mapsto e\_seek\_query), (T\_OP31p \mapsto e\_result\_query), (T\_OP32p \mapsto e\_not\_found)\}$

axm14: $EVT\_FIRST = \{e\_input\_query, e\_seek\_query, e\_result\_query, e\_not\_found)\}$

axm15: $causEEG = \{(causEE \rhd EVT\_FIRST) \cup causT1EE\}$

axm16: $causREG = \{(causRE \rhd EVT\_FIRST) \cup causT1RE\}$

axm17: $HcausG = \{(Hcaus \rhd EVT\_FIRST) \cup HcausT1$ **}**

axm18: $causG = \{causSYNC \cup causEEG \cup causREG \cup HcausG \cup causT2 \cup causT3\}$

axm19: $Begin = \{e\_connect\}$

axm20: $X\_OPERANDS = \{OP00, OP11, OP21, OP31, OP32\}$

axm21: $X\_WEIGHT = \{(e\_connect \mapsto 1), (e\_input\_query \mapsto 2), (e\_seek\_query \mapsto 3), (e\_result\_query \mapsto 3), (e\_transmit\_result \mapsto 3), (e\_not\_found \mapsto 3), (e\_display\_not\_found \mapsto 3), (r\_connect \mapsto 1), (r\_input\_query \mapsto 2), (r\_seek\_query \mapsto 3), (r\_result\_query \mapsto 3), (r\_transmit\_result \mapsto 3), (r\_not\_found \mapsto 3), (r\_display\_not\_found \mapsto 3), (T\_OP11p \mapsto 2), (T\_OP11n \mapsto 2), (T\_OP21p \mapsto 3), (T\_OP21n \mapsto 3), (T\_OP31p \mapsto 3), (T\_OP31n \mapsto 3), \}$

axm22: $EVT\_OP00 = \{e\_connect\}$

axm23: $EVT\_OP11 = \{e\_input\_query, r\_input\_query, T\_OP11p, T\_OP11n\}$

axm24: $EVT\_OP21 = \{e\_seek\_query, r\_seek\_query, T\_OP21p, T\_OP21n\}$

axm25: $EVT\_OP31 = \{e\_result\_query, r\_result\_query, e\_transmit\_result, r\_transmit\_result, T\_OP31p, T\_OP31n\}$

axm26: $EVT\_OP32 = \{e\_not\_found, r\_not\_found, e\_display\_not\_found, r\_display\_not\_found, T\_OP32p, T\_OP32n$**}**

END

FIGURE 6.10 – Quelques axiomes du contexte $CTX1P$

MACHINE **M1**
SEES **CTX1**
VARIABLES
state , current_lifeline, g1, g2, g3, g4
INVARIANTS
inv1: $state \in X\_EVTG \rightarrow \mathbb{N}$
inv2: $current\_lifeline \in X\_LIFELINES$
inv3: $(g1, g2, g3, g4) \in BOOL$
EVENTS
Initialisation
begin
act1: $state := X\_WEIGHT$
act2: $current\_instance := L1$
act3: $(g1, g2, g3, g4) :\in BOOL$
end
Event **e_connect** ⟨**ordinary**⟩ $\widehat{=}$
when
grd1: **...**
then
act1: **...**
end
Event **r_connect** ⟨**ordinary**⟩ $\widehat{=}$
when
grd1: **...**
then
act1: **...**
end
Event **T_OP11p** ⟨**ordinary**⟩ $\widehat{=}$
when
grd1: **...**
then
act1: **...**
end
Event **T_OP11n** ⟨**ordinary**⟩ $\widehat{=}$
when
grd1: **...**
then
act1: **...**
end
END

FIGURE 6.11 – Squelette de la machine B événementiel de l'étude de cas

Event T_OP21p ⟨**ordinary**⟩ $\widehat{=}$

when

grd1: $\forall EE \cdot ((EE \in EVT\_OP00 \wedge EE \in ((causG \setminus HcausG)^{-1}[\{T\_OP21p\}] \wedge (state(T\_OP21p) \ mod \ 3 = 0)) \Rightarrow ((state(EE) < state(T\_OP21p)/3)))$

grd2: $\forall EE \cdot ((EE \in EVT\_OP11 \wedge EE \in ((causG \setminus HcausG)^{-1}[\{T\_OP21p\}] \wedge (state(T\_OP21p) \ mod \ 3 = 0)) \Rightarrow ((state(EE)/2 < state(T\_OP21p)/3) \wedge (state(EE) \ mod \ 2 = 0)))$

grd3: $\forall EE \cdot ((EE \in (EVT\_OP21 \cup EVT\_OP31 \cup EVT\_OP32) \wedge EE \in HcausG^{-1}[\{T\_OP21p\}] \wedge (state(T\_OP21p) \ mod \ 3 \neq 0)) \Rightarrow ((state(EE) = state(T\_OP21p))))$

grd4: $state(T\_OP21p) \geq 1$

grd5: $g2 = TRUE$

then

act1: $state := state \Leftarrow \{(T\_OP21p \mapsto (state(T\_OP21p) - 1)), (T\_OP21n \mapsto (state(T\_OP21n) - 1))\}$

act2: $current\_lifeline := FCT\_l(T\_OP21p)$

act3: $g2 :\in BOOL$

end

FIGURE 6.12 – Implémentation de l'événement fictif positif de l'opérande $OP21$ du FC LOOP

Event T_OP21n ⟨**ordinary**⟩ $\widehat{=}$

when

grd1: $\forall EE \cdot ((EE \in EVT\_OP00 \wedge EE \in ((causG \setminus HcausG)^{-1}[\{T\_OP21n\}] \wedge (state(T\_OP21n) \ mod \ 3 = 0)) \Rightarrow ((state(EE) < state(T\_OP21n)/3)))$

grd2: $\forall EE \cdot ((EE \in EVT\_OP11 \wedge EE \in ((causG \setminus HcausG)^{-1}[\{T\_OP21n\}] \wedge (state(T\_OP21n) \ mod \ 3 = 0)) \Rightarrow ((state(EE)/2 < state(T\_OP21n)/3) \wedge (state(EE) \ mod \ 2 = 0)))$

grd3: $\forall EE \cdot ((EE \in (EVT\_OP21 \cup EVT\_OP31 \cup EVT\_OP32) \wedge EE \in HcausG^{-1}[\{T\_OP21n\}] \wedge (state(T\_OP21n) \ mod \ 3 \neq 0)) \Rightarrow ((state(EE) = state(T\_OP21n))))$

grd4: $state(T\_OP21n) \geq 1$

grd5: $g2 = FALSE$

then

act1: $state := state \Leftarrow \{x \mapsto y | x \in EVT\_OP21 \wedge y = (state(x) - state(T\_OP21n))\}$

act2: $current\_lifeline := FCT\_l(T\_OP21n)$

act3: $g2 :\in BOOL$

end

FIGURE 6.13 – Implémentation de l'événement fictif négatif de l'opérande $OP21$ du FC LOOP

Event T_OP31p ⟨**ordinary**⟩ $\widehat{=}$

when

grd1: $\forall EE \cdot ((EE \in EVT\_OP21 \ \wedge EE \in ((causG \setminus HcausG)^{-1}[\{T\_OP31p\}])) \Rightarrow (state(EE) < state(T\_OP31p)))$

grd2: $state(T\_OP31p) \geq 1$

grd3: $g3 = TRUE$

then

act1: $state := state \lhd (\{(T\_OP31p \mapsto (state(T\_OP31p) - 1)), (T\_OP31n \mapsto (state(T\_OP31n) - 1))\} \cup \{x \mapsto y | x \in EVT\_OP32 \ \wedge \ y = (state(x) - (state(T\_OP32n) - state(T\_OP31p) + 1))\})$

act2: $current\_lifeline := FCT\_l(T\_OP31p)$

act3: $g3 :\in BOOL$

end

FIGURE 6.14 – Implémentation de l'événement fictif positif de l'opérande $OP31$ du FC ALT

Event r_connect ⟨**ordinary**⟩ $\widehat{=}$

when

grd1: $\forall EE \cdot ((EE \in EVT\_OP00 \ \wedge EE \in ((causG \setminus HcausG)^{-1}[\{r\_connect\}]) \Rightarrow ((state(EE) < state(r\_connect)))$

grd2: $state(r\_connect) \geq 1$

then

act1: $state(r\_connect) := state(r\_connect) - 1$

act2: $current\_lifeline := FCT\_l(r\_connect)$

end

FIGURE 6.15 – Implémentation d'un événement normal

associés avec leur poids. L'état initial correspond à initialisation de la machine et l'état final est défini lorsque tous les événements sont consommés ou ignorés. Une transition d'un état i à un état j correspond à l'occurrence d'un événement. La Figure 6.18 représente une vue partielle du diagramme état transition généré par ProB. ProB offre aussi la possibilité de vérifier si une trace est valide ou non en proposant une interface permettant de suivre l'exécution des événements pas par pas. Les Figures 6.26 et 6.27 illustrent ce type d'exécution, les événements qui ne sont pas déclenchés ou qui sont déclenchés sont marqués en rouge et les événements habilités sont marqués en vert.

6.3.3.3 Vérification de quelques propriétés temporelles avec ProB

Nous proposons de vérifier quelques propriétés temporelles avec ProB. Nous choisissons de spécifier des propriétés qui sont intrinsèques au système étudié (Figure 6.19) et des propriétés générales pouvant être vérifiées pour n'importe quelle spécification associée à n'importe quel DS (Figure 6.20 et Figure 6.21). $P1$: nous exprimons que l'événement r_seek_query va être suivi

Event T_OP31n ⟨**ordinary**⟩ $\widehat{=}$

when

grd1: $\forall EE \cdot ((EE \in EVT\_OP21 \ \wedge EE \in ((causG \setminus HcausG)^{-1}[\{T\_OP31n\}])) \Rightarrow (state(EE) < state(T\_OP31n)))$

grd2: $state(T\_OP31n) \geq 1$

grd3: $g3 = FALSE$

then

act1: $state := state \Leftarrow \{x \mapsto y | x \in EVT\_OP31 \ \wedge y = (state(x) - 1)\}$

act2: $current\_lifeline := FCT\_l(T\_OP31n)$

act3: $g3 :\in BOOL$

end

FIGURE 6.16 – Implémentation de l'événement fictif négatif de l'opérande $OP31$ du FC ALT

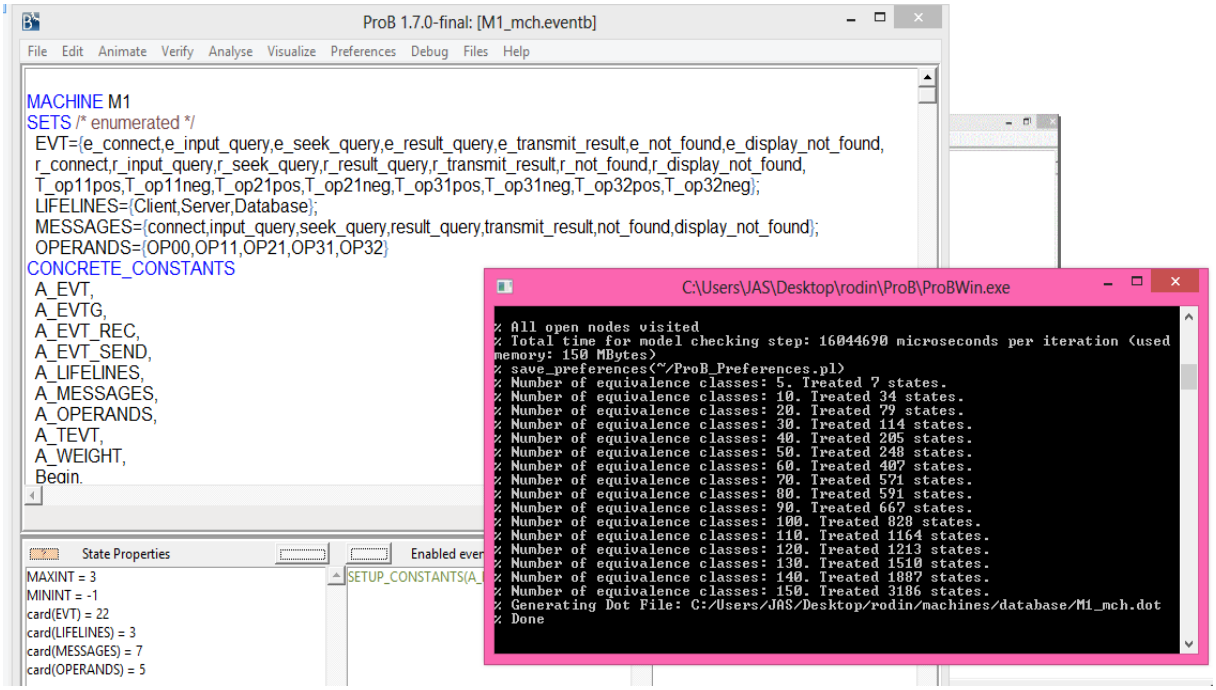

FIGURE 6.17 – Interface ProB de l'étude de cas

soit par l'événement $r\_transmit\_result$, soit par l'événement $r\_display\_not\_found$ ($P3$).
$P2$: nous exprimons que chaque événement émis va être reçu.
$P3$: nous exprimons que chaque événement reçu a été envoyé (i.e chaque message reçu est précédé par un message envoyé) avec la formule ($P2$).
Les formules LTL(e) des propriétés $P1$, $P2$ et $P3$ sont respectivement représentées dans les Figures 6.19, 6.20 et 6.21. Le résultat de la vérification des propriétés $P2$ et $P3$ est représenté dans la Figure 6.22. Le résultat de la vérification de la propriété est $P1$ est représenté dans la Figure 6.23. L'explorateur de modèles ProB vérifie la formule qui exprime la propriété désirée. Il affiche 3 types statuts : *i)* soit la formule n'est pas encore vérifiée ; *ii)* soit elle est vraie pour tous les chemins c'est le cas des propriétés $P2$ et $P3$; *iii)* soit elle est fausse, dans ce cas l'explorateur de modèles fournit un contre-exemple (i.e. le chemin qui ne satisfait pas la

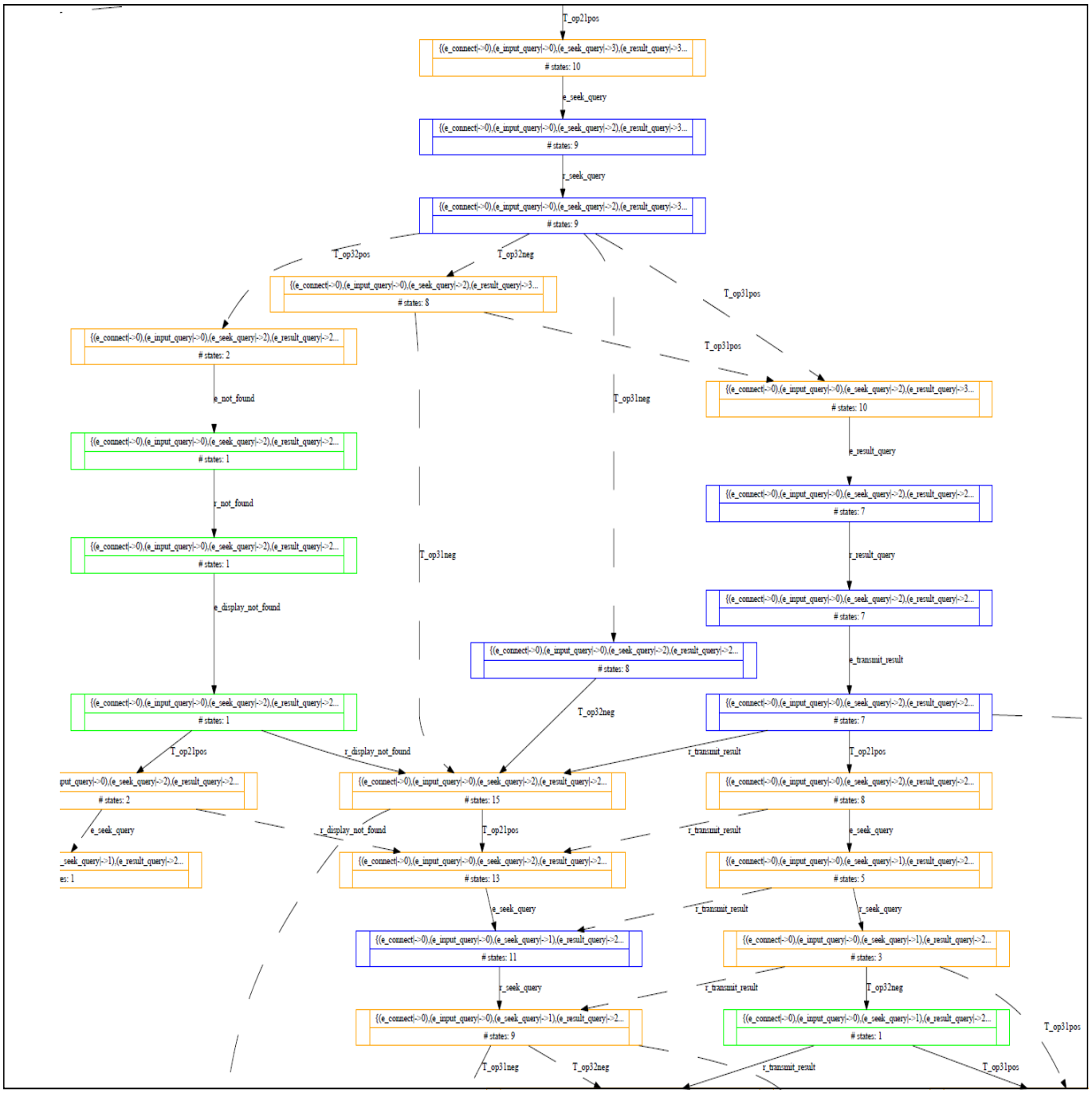

FIGURE 6.18 – Visualisation d'une vue partielle du diagramme état transition avec une projection sur la variable état avec l'explorateur de modèles ProB

formule), c'est le cas de la formule de la propriété $P1$. Le contre exemple est représenté sous forme d'un chemin fini menant à un état de blocage (état coloré en rouge). Ce résultat est logique puisque si toutes les gardes du fragment combiné ALT sont fausses on n'obtient pas forcément après l'occurrence de l'événement r_seek_query ni l'événement $r\_transmit\_result$ ni l'événement $r\_display\_not\_found$.

Statistiques de couverture (*coverage statistics*)	
Nombre total de nœud	3330
Nombre total de transitions	6653
Statistiques sur les nœuds	
Bloquées	0
Violation invariant	0
Invariant non vérifié	0
Nœuds ouverts [11]	0
Transitions explorées mais pas toutes sont calculées	1
Live [12]	3330
Total	3330

TABLE 6.5 – Quelques statistiques fournies par ProB

P1 : $G(e(r\_seek\_query) \Rightarrow F(e(r\_transmit\_result)) \vee e(r\_display\_not\_found)))$

FIGURE 6.19 – Formule LTL[e] de la propriété $P1$

P2 : $G(e(e\_seek\_query) \Rightarrow F(e(r\_seek\_query))) \;\&$
$G(e(e\_connect) \Rightarrow F(e(r\_connect))) \;\&$
$G\ (e(e\_input\_query) \Rightarrow F(e(r\_input\_query))) \;\&$
$G(e(e\_result\_query) \Rightarrow F(e(r\_result\_query))) \;\&$
$G(e(e\_transmit\_result) \Rightarrow F(e(r\_transmit\_result))) \;\&$
$G(e(e\_not\_found) \Rightarrow F(e(r\_not\_found))) \;\&$
$G(e(e\_display\_not\_found) \Rightarrow F(e(r\_display\_not\_found)))$

FIGURE 6.20 – Formule LTL[e] de la propriété $P2$

P3 : $G(e(r\_seek\_query) \Rightarrow O(e(e\_seek\_query))) \;\&$
$G(e(r\_connect) \Rightarrow O(e(e\_connect))) \;\&$
$G(e(r\_input\_query) \Rightarrow O(e(e\_input\_query))) \;\&$
$G(e(r\_result\_query) \Rightarrow O(e(e\_result\_query))) \;\&$
$G(e(r\_transmit\_result) \Rightarrow O(e(e\_transmit\_result))) \;\&$
$G(e(r\_not\_found) \Rightarrow O(e(e\_not\_found))) \;\&$
$G(e(r\_display\_not\_found) \Rightarrow O(e(e\_display\_not\_found)))$

FIGURE 6.21 – Formule LTL[e] de la propriété $P3$

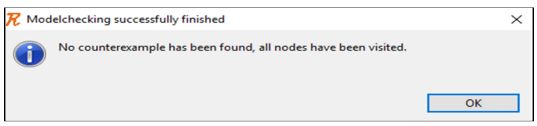

FIGURE 6.22 – Image écran du résultat de la vérification des propriétés $P2$ et $P3$

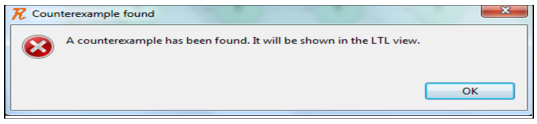

FIGURE 6.23 – Image écran du résultat de vérification de la propriété $P1$

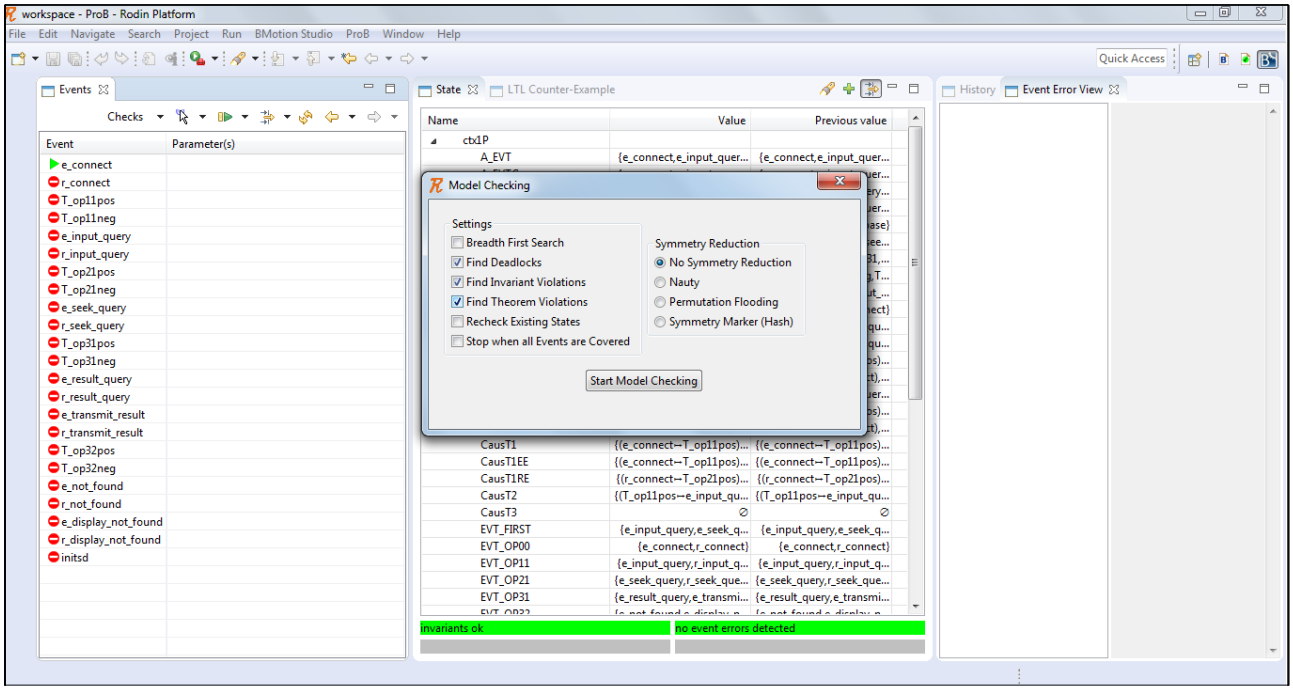

FIGURE 6.24 – L'explorateur de modèles

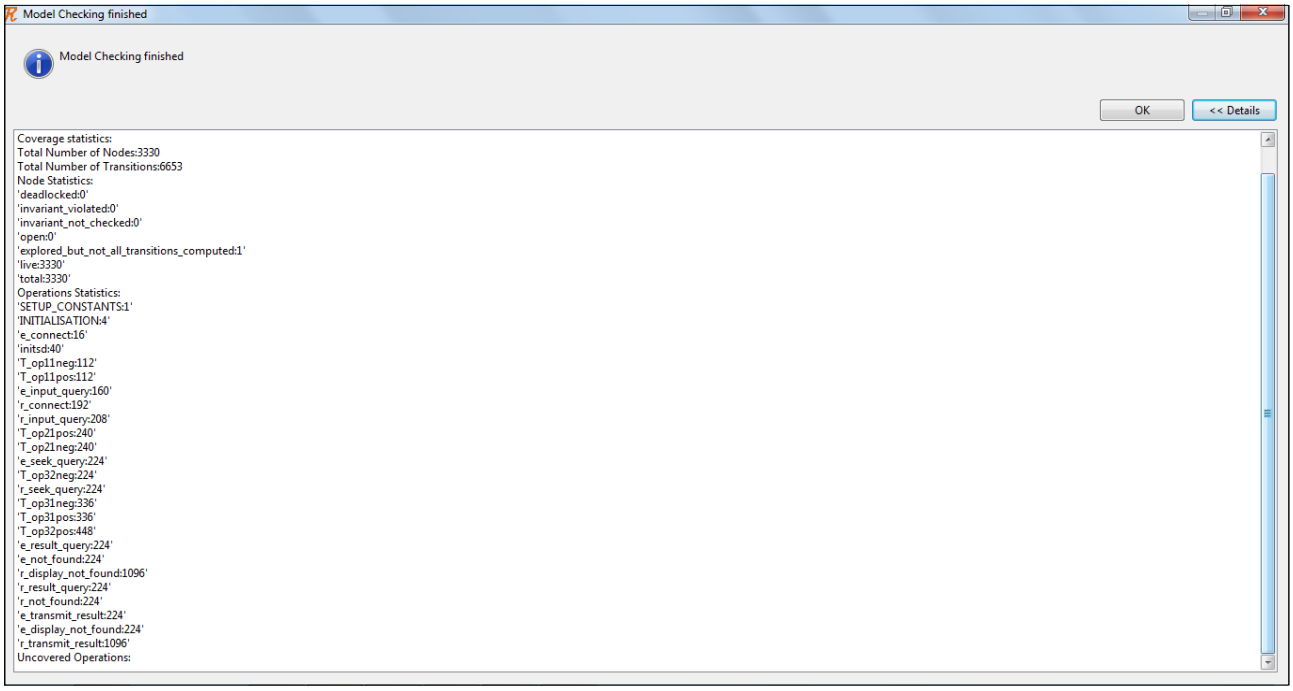

FIGURE 6.25 – Résultat fourni par l'explorateur de modèles

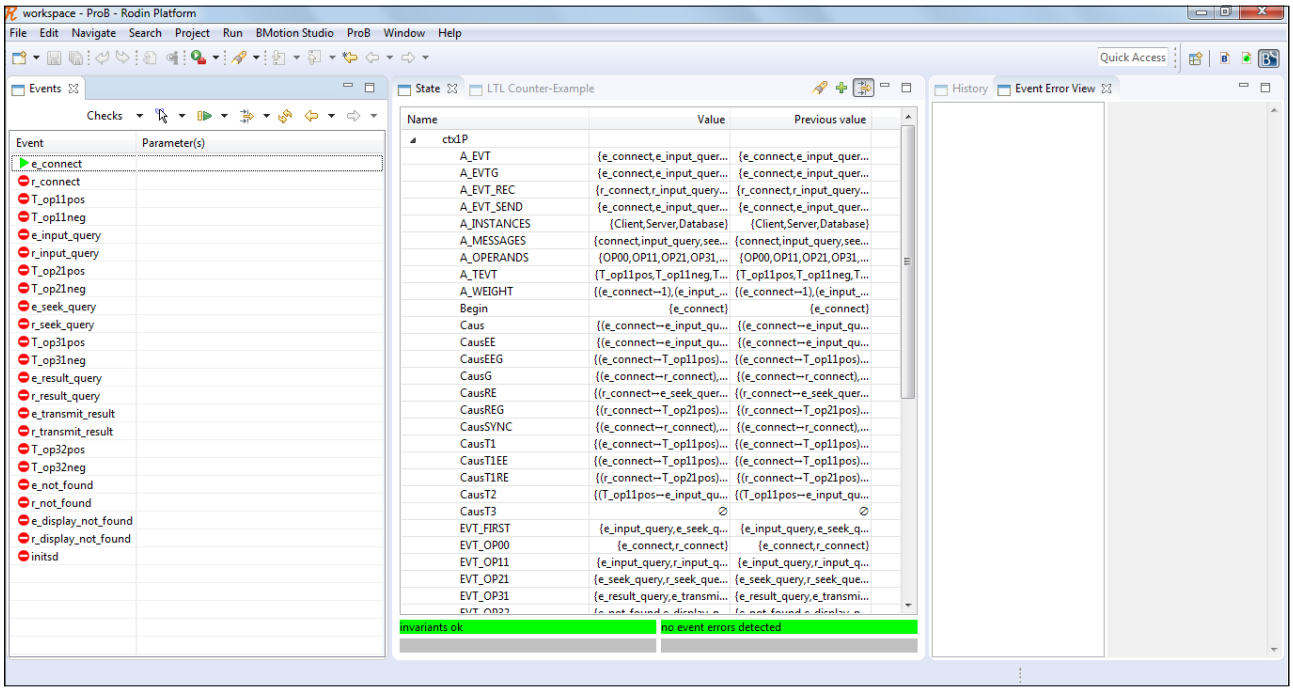

FIGURE 6.26 – Exécution pas à pas : pas1

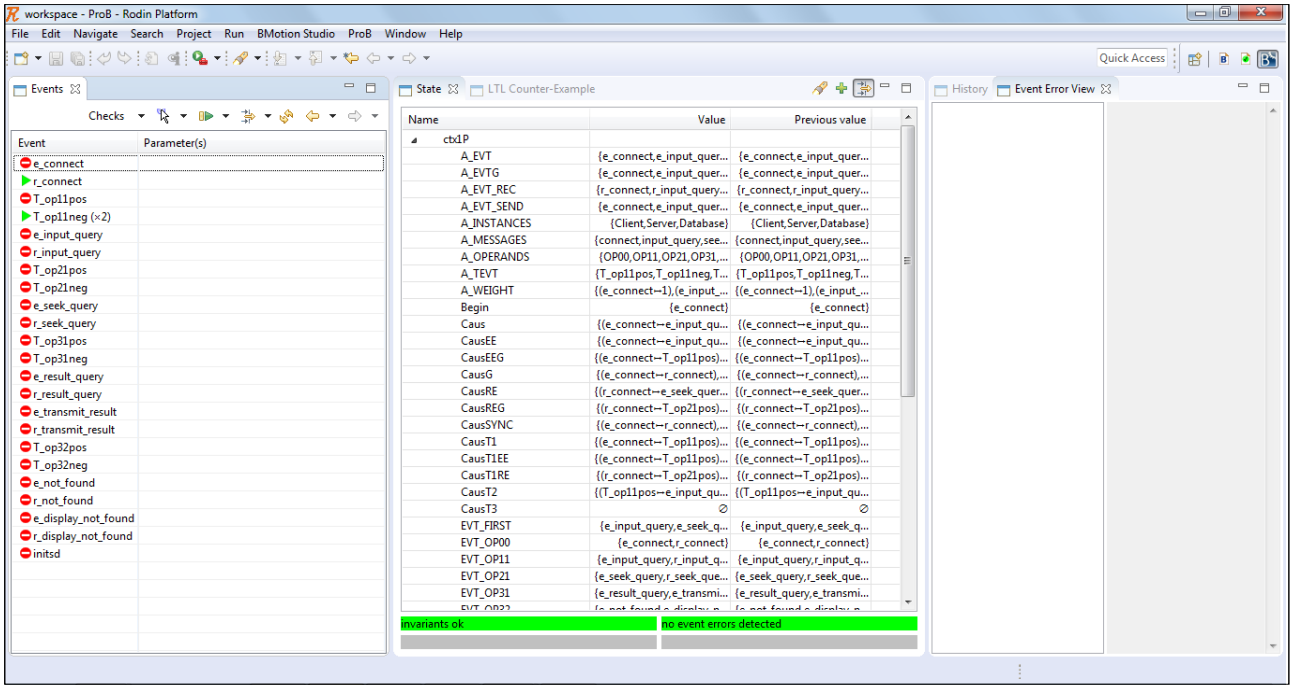

FIGURE 6.27 – Exécution pas à pas : pas2

6.4 Conclusion

À travers l'implémentation de plusieurs diagrammes de séquence avec FC imbriqués, nous avons abouti à la proposition d'un modèle de traduction générique des diagrammes de séquence d'UML2.X vers des spécifications en B événementiel. Ces implémentations nous permettent de raisonner sur la vérification de la relation de raffinement entre deux diagrammes de séquence modélisant des comportements complexes de systèmes distribués, qui fera l'objet du chapitre

suivant.

Chapitre 7

Implémentation de la relation de raffinement des diagrammes de séquence d'UML2.X en B événementiel

Sommaire

7.1 Introduction

La méthode B événementiel supporte explicitement les deux interprétations du raffinement qui sont :
• le raffinement vu comme un processus qui permet d'introduire des détails aux descriptions d'un programme et
• la description de sa solution.
Le raffinement en B événementiel permet de raffiner les structures de données et de rajouter des détails avec la définition de nouveaux événements. Le raffinement des états s'exprime à l'aide d'un invariant de collage. Le raffinement au niveau des événements se traduit par un renforcement des gardes et par la préservation de l'invariant de collage.

La relation de raffinement B événementiel est basée sur la traduction d'alphabet qui nécessite une correspondance explicite entre les composants des DS, ce qui permet de détecter les erreurs de modélisation et par conséquent d'apporter les corrections nécessaires. En outre la relation de raffinement permet la vérification de la terminaison des nouveaux événements (leur non-divergence).

7.2 Modèle générique du processus de raffinement

Dans le Chapitre 6, nous avons proposé un modèle de traduction des diagrammes de séquence d'UML2.X qui contiennent des FC imbriqués. Nous étendons ce modèle pour proposer un modèle générique permettant de vérifier la relation de raffinement entre deux diagrammes de séquence.
Rappelons que le modèle de traduction d'un diagramme de séquence DSi comprend les composants B suivants :

— un contexte CTX qui contient tous les composants du DS considéré,
— un contexte $CTXi$ qui contient les axiomes exprimant des propriétés sur les composants des DS qui sont modélisés par des constantes,
— un contexte $CTXiP$ qui contient les valeurs concrètes ou réelles des composants du DS considéré et
— une machine B qui contient les variables d'états, les invariants et les événements.

Étant donné deux diagrammes de séquence, $DS1$ et $DS2$, tels que $DS2$ est supposé être le raffinement du $DS1$; le modèle générique du processus de raffinement (voir Figure 7.1) comprend d'une part les spécifications en B événementiel représentant la traduction de chaque DS de façon indépendante, d'autre part de nouveaux composants (nouveaux contextes) qui permettent de faire le lien entre les spécifications relatives aux DS considérés encodés. Les nouveaux composants correspondent aux nouveaux contextes que nous nommons $CTX2$ et $CTX2P$. Dans le contexte $CTX2$, qui est représenté par la Figure 7.2, nous fournissons le typage des nouvelles variables introduites (axiomes $1, 2, 3$ *et* 5) et nous y exprimons des propriétés (axiomes $4, 6, 7, 8$ *et* 9). En outre, nous exprimons quelques règles du raffinement structurel que nous avons déjà fixées :
• dans l'axiome 10 nous exprimons la règle $R3$ du raffinement,
• dans l'axiome 11 nous exprimons les règles $R1$ et $R2$ du raffinement,
• dans l'axiome 12 nous exprimons la correspondance entre les lignes de vie des DS considérés (abstrait et raffiné).

Dans le contexte $CTX2P$ nous donnons les valeurs concrètes des nouveaux composants introduits (les nouvelles lignes de vie, les nouveaux messages, événements et opérandes). Nous fournissons aussi la correspondance concrète entre les lignes de vie des deux DS considérés.

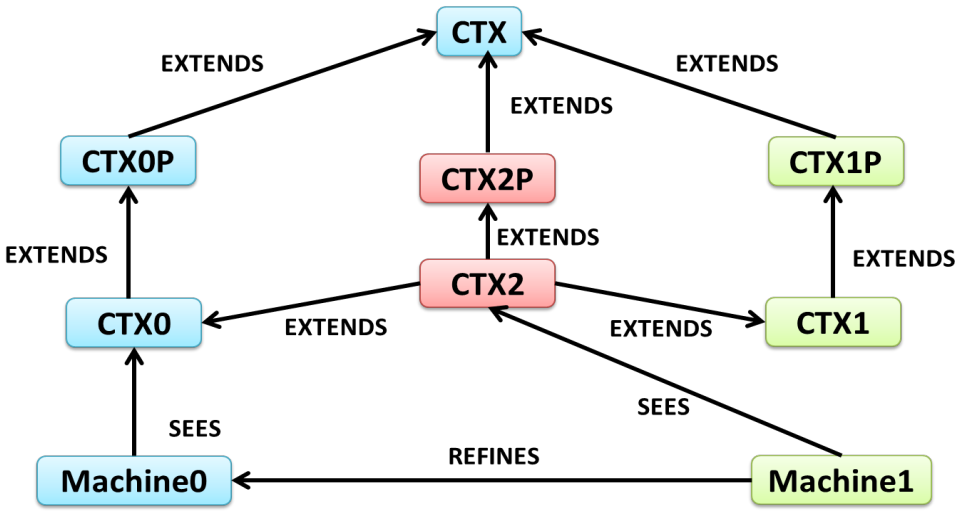

FIGURE 7.1 – L'architecture générique de l'implémentation du processus de raffinement

Dans la machine raffinée (dont nous représentons le squelette par la Figure 7.3), nous retrouvons les mêmes variables que celles de la machine abstraite auxquelles nous pouvons rajouter de nouvelles variables qui sont intrinsèques au DS raffiné. Dans l'invariant nous exprimons, en plus du typage de chaque variable, l'invariant du collage qui permet de lier les variables abstraites

aux variables concrètes. L'invariant de collage exprime la correspondance des états relatifs au DS abstrait et concret. Nous définissons également le variant qui permet de garantir la non divergence des nouveaux événements introduits. Les événements de la machine raffinée sont les événements de la machine abstraite auxquels quelques nouveaux événements peuvent être ajoutés. Le statut de chaque nouvel événement est marqué *convergent* (ce qui veut dire qu'il ne prend pas le contrôle indéfiniment). La vérification de la relation de raffinement entre deux diagrammes de séquences se fait avec les obligations de preuves dont quelques unes sont prouvées automatiquement et les autres sont prouvées interactivement.
La vérification de la terminaison des nouveaux événements est assurée en prouvant que le variant est décrémenté pour chaque occurrence d'un nouvel événement (le variant doit rester strictement supérieur à 0).
En prouvant la terminaison des nouveaux événements, nous satisfaisons la règle $R5$ du raffinement (Chapitre 5 Section 5.4.1).
Une fois que toutes les obligations de preuves ont été satisfaites nous pouvons affirmer que nous avons un raffinement structurel et sémantique correct.

CONTEXT **CTX2**
EXTENDS **CTX0, CTX1, CTX2P**
AXIOMS

axm1: $New\_LIFELINES1 \subseteq LIFELINES$
axm2: $New1\_messages \subseteq MSG$
axm3: $New1\_EVT \subseteq EVT\}$
axm4: $New1\_EVT \cap A\_EVT = \varnothing$
axm5: $LIFELINES1\_NOT\_REFINED \subseteq A\_LIFELINES$
axm6: $R1\_LIFELINES = LIFELINES1\_NOT\_REFINED \cup New\_LIFELINES1$
axm7: $LIFELINES1\_NOT\_REFINED \cap New\_LIFELINES1 = \varnothing\}$
axm8: $R1\_MSG = A\_MSG \cup New1\_messages$
axm9: $R1\_EVT = A\_EVT \cup New1\_EVT$
axm10: $ran(New1\_EVT \lhd FCT\_l\_r1) = New\_LIFELINE1S$
axm11: $FCT\_l = ((A\_EVT \lhd FCT\_l\_r1); FCT\_LIFELINES)$
axm12: $FCT\_LIFELINES \in R1\_LIFELINES \rightarrow A\_LIFELINES$

END

FIGURE 7.2 – Partie du code B du contexte $CTX2$

Nous appliquons notre approche de vérification formelle en B événementiel de la relation de raffinement entre les spécifications relatives aux diagrammes de séquence encodés DS modélisant des comportements des systèmes distribués à travers les expérimentations que nous présentons dans ce qui suit.

7.3 Expérimentation avec le prouveur des théorèmes Rodin

7.3.1 Modélisation par les diagrammes de séquence

Nous modélisons les interactions dans un restaurant avec des diagrammes de séquence d'UML2.X (voir Figure 7.4). Le système restaurant possède des composants qui sont distribués et indépendants : le client modélisé par la ligne de vie *Client*, le chef des serveurs modélisé par la ligne de

```
MACHINE M2
REFINES M1
SEES CTX2
VARIABLES
    V1
    ...
INVARIANTS
    inv1: ...
    inv2: ...
VARIANT
    ...
EVENTS
Initialisation
  begin
      act1: ...
      act2: ...
  end
Event evt1 ⟨ordinary⟩ ≙
refines evt1
  when
      grd1:
  then
      act1:
  end
Event T_OPXpos ⟨ordinary⟩ ≙
refines T_OPXpos
  when
      grd1:
  then
      act1:
  end
Event evt3 ⟨convergent⟩ ≙
  when
      grd1:
  then
      act1:
  end
END
```

FIGURE 7.3 – Squelette de la machine raffinée

vie *H_waiter*, la cuisine qui est modélisée par la ligne de vie *Kitchen* et le barman modélisé par

la ligne de vie *Bar_k*.

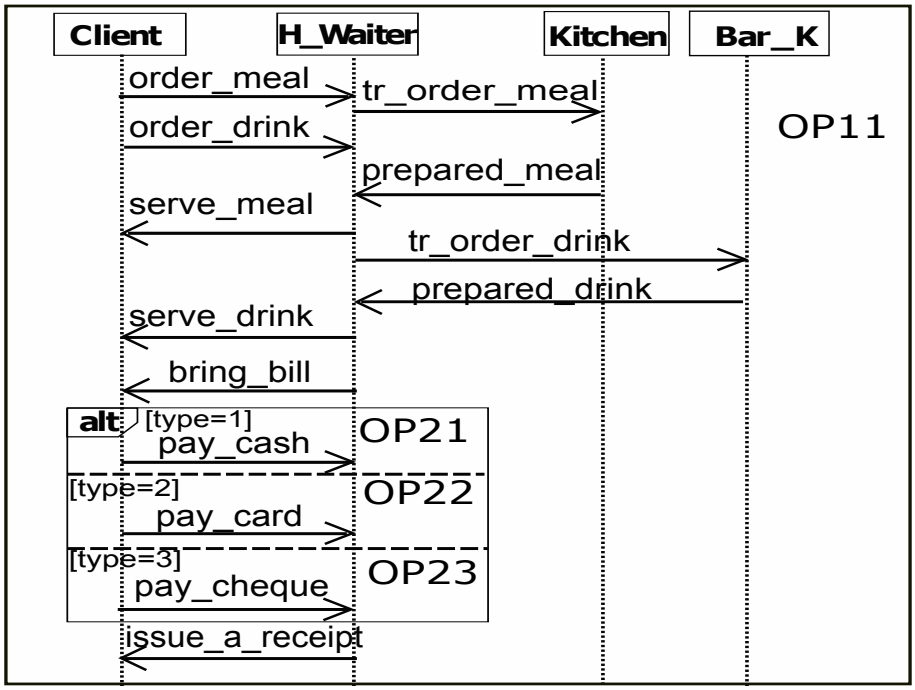

FIGURE 7.4 – DS0 : interactions dans un restaurant

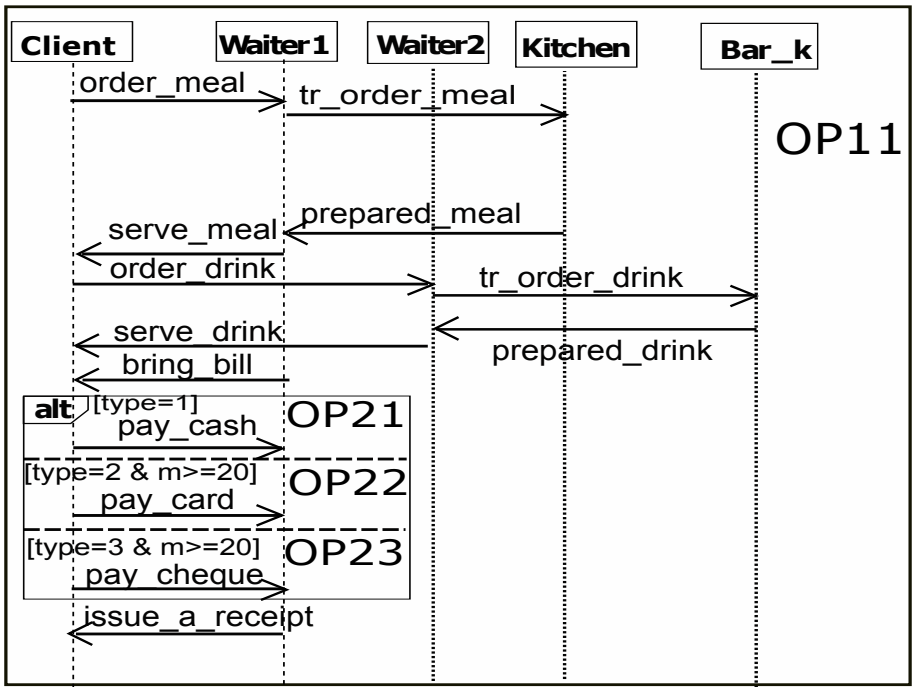

FIGURE 7.5 – DS1 : premier raffinement de DS0 de la Figure 7.4

Le *Client* ordonne au *H_waiter* le plat principal ainsi que la ou les boissons. Le *H_waiter* transmet les ordres repectivement à la cuisine *Kitchen* et au barman *Bar_k*. Une fois que les ordres sont prêts, la cuisine et le barman alertent le *H_waiter* pour les servir. Ensuite le *H_waiter* ramène l'addition. Le client peut régler la facture plusieurs façons : soit en espèces, soit par chèques ou encore par carte bancaire. Une fois que la facture est payée, le *H_waiter* délivre un reçu au client.

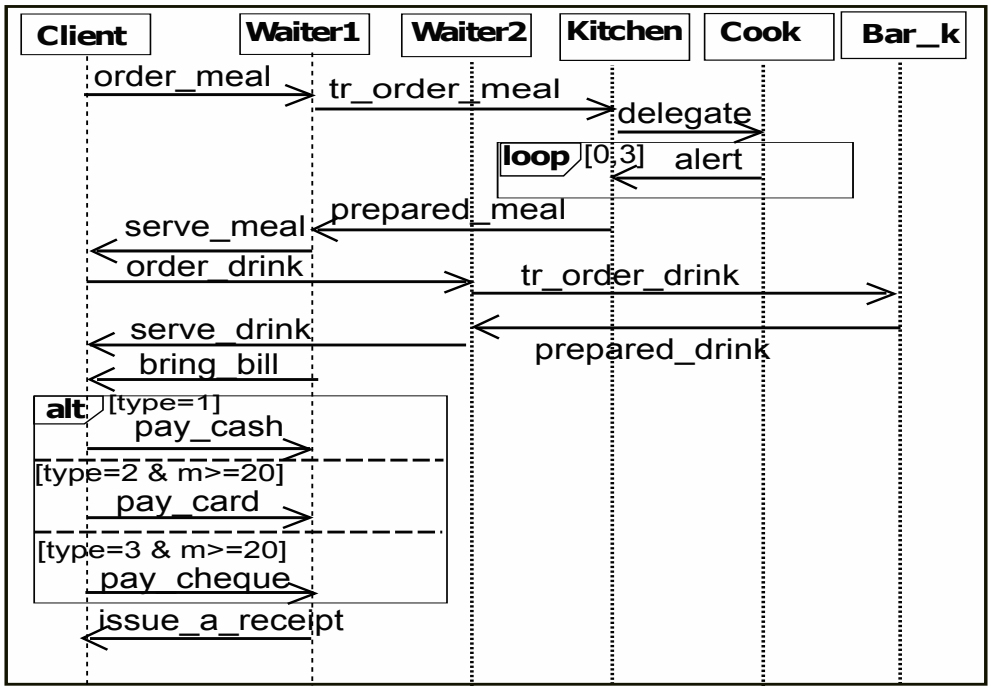

FIGURE 7.6 – DS2 : deuxième raffinement de DS0 de la Figure 7.4

7.3.1.1 Premier raffinement

Un raffinement possible du DS représenté par la Figure 7.4 est donné par le DS représenté par la Figure 7.5. La ligne de vie abstraite *H_waiter* est substituée par les nouvelles lignes de vie $Waiter1$ et $Waiter2$. Les événements de la ligne de vie *H_waiter* sont distribués entre les nouvelles lignes de vie.

7.3.1.2 Deuxième raffinement

Un deuxième raffinement du DS de la Figure 7.5 est donné par le DS représenté par la Figure 7.6. Le comportement interne de la ligne de vie $Kitchen$ est détaillé : une nouvelle sous ligne de vie *Cook* est ajoutée dans le DS2. La ligne de vie $Kitchen$ délègue la tâche de préparation du plat principal à la nouvelle ligne de vie *Cook*. Une fois que le repas est prêt, le *Cook* alerte le $Kitchen$ au plus 3 fois.

7.3.2 Spécifications en B événementiel

Nous optons pour la fusion des deux raffinements dans une même spécification raffinée. Ainsi nous considérons le DS de la Figure 7.4 comme DS abstrait et le DS de la Figure 7.6 comme son raffinement. Comme nous l'avons mentionné, nous traduisons indépendamment chaque DS vers des spécifications B événementiel.
La Figure 7.7 représente le contexte contenant tous les composants des deux DS considérés. Le modèle B événementiel de chaque DS est composé de deux contextes et d'une machine.
Dans ce qui suit fournissons quelques parties des spécifications B. Les Figures 7.8 et 7.9 représentent respectivement une partie du code B des spécifications de la machine abstraite et de la machine raffinée.
La Figure 7.10 représente le nouveau contexte CTX2P qui est requis essentiellement pour faire la correspondance de quelques composants des DS considérés. En effet :
◇ dans les trois premiers axiomes, nous exprimons respectivement les nouvelles lignes de vie, les nouveaux messages ainsi que les nouveaux événements qui sont introduits dans le DS raffiné.
◇ l'axiome 4 représente la ligne de vie qui n'a pas été raffinée,

$\diamond$ dans le cinquième axiome, nous exprimons le mappage entre les lignes de vie du DS abstrait et les lignes de vie du DS raffiné.
Dans les Figures 7.11, 7.12 et 7.13, nous représentons l'implémentation de quelques événements de l'étude de cas. La Figure 7.11 représente l'implémentation de l'événement abstrait *R_order_meal*. La Figure 7.11 représente l'implémentation du raffinement de l'événement abstrait *R_order_meal*. La Figure 7.13 représente l'implémentation d'un nouvel événement introduit *E_alert*.

Le raffinement B est prouvé à l'aide des obligations de preuve qui sont fournies par le prouveur de théorèmes du Rodin. Certaines OP nécessitent une intervention pour les prouver cas par cas manuellement.

7.3.2.1 Échec de quelques obligations de preuves

Quelques obligations de preuves du raffinement échouent. Ils correspondent aux problèmes que nous avons cité dans le Chapitre 5, Section 5.4.3.

Illustration : une première obligation de preuve concerne la perte de l'ordre des événements abstraits dans le DS raffiné. Ce sont les événements *E_serve_drink* et *E_bring_bill*, qui appartiennent à la même ligne de vie et qui sont ordonnés dans le diagramme de séquence DS1 selon la relation $<_{EE}$. Cependant cet ordre est perdu dans le diagramme de séquence raffiné DS2 puisqu'ils sont distribués entre les nouvelles lignes de vie $waiter1$ et $waiter2$ qui substituent la ligne de vie abstraite *H_waiter*. L'événement *E_bring_bill* peut se déclencher avant ou après l'événement *E_serve_drink* ; par conséquent de nouvelles traces qui n'ont pas leurs contreparties peuvent être générées dans DS2.

Pour corriger ce problème nous proposons de restituer les relations de précédence perdues. Cette solution est proche d'une solution donnée par l'approche de [Ohl06] qui propose d'ajouter des messages entre les événements distribués entre les nouvelles lignes de vie.
L'échec d'une seconde obligation de preuve est causé par le renforcement de la seconde et la troisième garde du FC ALT qui fait perdre quelques cas. En effet le serveur peut délivrer un reçu bien que le consommateur n'a pas payé sa facture.

7.3.2.2 Vérification de la terminaison des nouveaux événements introduits

Chaque nouvel événement possède un statut convergent. Pour le DS raffiné, nous avons 4 nouveaux événements : *E_alert* et *R_alert* qui appartiennent à une opérande LOOP et qui peuvent se déclencher au maximum 3 fois et les événements *E_delegate* et *R_delegate*.
Nous définissons le variant qui est l'addition des états de tous les nouveaux événements introduits. $(state\_r1(E\_alert) + state\_r1(R\_alert) + state\_r1(E\_delegate) + state\_r1(R\_delegate) + state\_r1(T\_OP31pos) + state\_r1(T\_OP31neg))$ Ce variant est décrémenté à chaque fois où un nouvel événement est déclenché. Le variant ne s'annule jamais et il garantit que les nouveaux événements ne divergent pas.

La vérification de la relation de raffinement avec le prouveur des théorèmes n'est pas évidente et nécessite un temps assez long pour la correction de preuves qui sont traitées une par une. Cependant avec l'explorateur des modèles ProB le temps de vérification est très rapide. Par conséquent, dans les autres expérimentations que nous avons conduit, nous avons utilisé le ProB pour la vérification de la relation de raffinement des spécifications relatives aux DS considérés encodés.

CONTEXT CTX
SETS
LIFELINES
MSG
EVT
CONSTANTS
client, H_waiter, kitchen, bar_k....
ordermeal, servemeal, orderdrink...
E_ordermeal, R_ordermeal, E_servemeal, R_servemeal...
T_op31pos, T_op31neg, T_op21pos, T_op21neg, InitSD...
AXIOMS
axm1: $partition(LIFELINES, \{client\}, \{h\_waiter\}, \{kitchen\}, \{bar\}, ...)$
axm2: $partition(MSG, \{ordermeal\}, \{servemeal\}, \{orderdrink\}, \{servedrink\}...)$
axm3: $partition(EVT, \{E\_ordermeal\}, \{R\_ordermeal\}, \{E\_servemeal\}...)$
END

FIGURE 7.7 – Partie du code B du contexte CTX

MACHINE M0
SEES CTX0
VARIABLES
state
current_lifeline
type_payment
INVARIANTS
inv1: $state \in A\_EVTG \rightarrow \mathbb{Z}$
inv2: $current\_lifeline \in A\_LIFELINE$
inv3: $type\_payment \in \mathbb{N}$
EVENTS
Initialisation
begin
act1: $state := A\_EVTG \times \{1\}$
act2: $current\_lifeline := client$
act3: $type\_payment := 0$
end
END

FIGURE 7.8 – Partie du code en B de la machine abstraite

7.4 Expérimentation avec l'explorateur des modèles ProB

Pour tester la relation de raffinement supportée par ProB, nous avons considéré dans un premier temps le raffinement de diagrammes de séquence basiques en considérant d'abord des DS ayant les mêmes événements puis des DS n'ayant pas les mêmes événements.

MACHINE M1
REFINES M0
SEES CTX2
VARIABLES

state_r1

current_lifeline_r1

type_payment

taking_into_account

INVARIANTS

inv1: $state\_r1 \in R1\_EVTG \rightarrow \mathbb{Z}$

inv2: $current\_lifeline\_r1 \in LIFELINES$

inv3: $current\_lifeline\_r1 \in R1\_LIFELINES$

inv4: $current\_lifeline = FCT1\_LIFELINES(current\_lifeline\_r1)$

inv5: $New1\_EVT \lhd state\_r1 = state$

inv6: $type\_payment \in \mathbb{N}$

act7: $taking\_into\_account \in BOOL$

VARIANT

$(state\_r1(E\_alert) + state\_r1(R\_alert) + state\_r1(E\_delegate) + state\_r1(R\_delegate) + state\_r1(T\_OP31pos) + state\_r1(T\_OP31neg))$

EVENTS
Initialisation

begin

act1: $type\_payment := 0$

act2: $taking\_into\_account := FALSE$

act3: $state\_r1 := ((R1\_EVTG \setminus EVT\_ROP31) \times \{1\}) \cup (EVT\_ROP31 \times \{Ni\})$

act4: $current\_lifeline\_r1 := client$

end

END

FIGURE 7.9 – Partie du code en B de la machine raffinée

Puis nous avons utilisé le ProB pour apporter des réponses, que nous avons abordé dans le Chapitre raffinement 5, pour des cas spécifiques de raffinement de DS comportant des FC.

7.4.1 Vérification de la relation de raffinement pour des diagrammes de séquence basiques

• Nous avons vérifié qu'une machine raffine elle même. Par conséquent, pour deux machines ayant le même alphabet (mêmes événements) et les mêmes traces, nous réussissons à montrer qu'il existe une relation de simulation et qu'il y a un raffinement de traces.
• Nous avons considéré deux machines identiques. Dans la machine concrète, nous avons renommé les événements. ProB prouve qu'il y a une relation de simulation. Cependant il y a un échec pour la preuve de l'existence d'une relation de raffinement de traces. Donc la relation de raffinement supportée par le ProB n'est pas substitutive.

• Nous avons considéré deux DS ayant les mêmes événements cependant le DS concret contient

CONTEXT **CTX2P**
EXTENDS **CTX**
CONSTANTS

New1_messages
New_LIFELINES1
LIFELINES1_NOT_REFINED
FCT1_LIFELINES
New1_EVT

AXIOMS

axm1: $New1\_messages = \{delegate, alert\}$
axm2: $New\_LIFELINES1 = \{cook, waiter1, waiter2\}$
axm3: $LIFELINES1\_NOT\_REFINED = \{client\}$
axm4: $FCT1\_LIFELINES = \{(client \mapsto client), (waiter1 \mapsto h\_waiter), (waiter2 \mapsto h\_waiter), (kitchen \mapsto kitchen), (cook \mapsto kitchen), (bar\_k \mapsto bar\_k)\}$
axm5: $New1\_EVT = \{E\_delegate, E\_alert, R\_delegate, R\_alert, T\_OP31pos, T\_OP31neg\}$

END

FIGURE 7.10 – Code B du contexte CTX2P

Event **R_ordermeal** ⟨**ordinary**⟩ $\widehat{=}$

when

grd1: $\forall EE \cdot ((EE \in EVT \wedge EE \in (Caus^{-1}[\{R\_ordermeal\}])) \Rightarrow (state(EE) < state(R\_ordermeal)))$
grd2: $state(R\_ordermeal) \geq 1$

then

act1: $state(R\_ordermeal) := state(R\_ordermeal) - 1$
act2: $current\_lifeline := FCT\_l(R\_ordermeal)$

end

FIGURE 7.11 – Code B d'un événement abstrait

moins de traces. ProB prouve qu'il y a une relation de simulation ainsi qu'une relation de raffinement de traces entre les machines relatives aux DS considérés.

Illustration : considérons le DS abstrait représenté par la Figure 7.14. Soit le DS concret représenté par la Figure 7.15. Dans le DS concret nous avons déplacé le message m de telle sorte que nous obtenons une nouvelle relation de précédence entre les événement $?m$ et $!m3$ sans influer sur les relations de précédence qui sont déjà existantes. Par conséquent le DS concret contient moins de traces que le DS abstrait. Pour tester la relation de raffinement entre les spécifications concrète et abstraite des DS encodés, nous vérifions d'abord la machine abstraite l'explorateur des modèles ProB. Ce denier génère un digramme état-transition. Ensuite, nous sauvegardons ces états afin de pouvoir vérifier le raffinement (Figure 7.16). Puis, nous vérifions la machine concrète par l'explorateur des modèles ProB. En générant le digramme état-transition relatif à la machine concrète.

Event **R_ordermeal** ⟨**ordinary**⟩ $\widehat{=}$
refines **R_ordermeal**
when
grd1: $\forall EE \cdot ((EE \in EVT \wedge EE \in (R1\_Caus^{-1}[\{R\_ordermeal\}])) \Rightarrow (state\_r1(EE) < state\_r1(R\_ordermeal)))$
grd2: $state\_r1(R\_ordermeal) \geq 1$
then
act1: $state\_r1(R\_ordermeal) := state\_r1(R\_ordermeal) - 1$
act2: $current\_lifeline\_r1 := FCT\_l\_r1(R\_ordermeal)$
end

FIGURE 7.12 – Code B du raffinement d'un événement abstrait

Event **E_alert** ⟨**convergent**⟩ $\widehat{=}$
when
grd1: $\forall EE \cdot ((EE \in EVT \wedge EE \in (R1\_Caus^{-1}[\{E\_alert\}])) \Rightarrow (state\_r1(EE) < state\_r1(E\_alert)))$
grd2: $state\_r1(E\_alert) \geq 1$
then
act1: $state\_r1(E\_alert) := state\_r1(E\_alert) - 1$
act2: $current\_lifeline\_r1 := FCT\_l\_r1(E\_alert)$
act3: $taking\_into\_account :\in \{TRUE, FALSE\}$
end

FIGURE 7.13 – Code B d'un nouvel événement

ProB prouve qu'il existe une relation de simulation. Nous vérifions également qu'il a une relation de raffinement de traces entre les machines considérées (Figure 7.17).

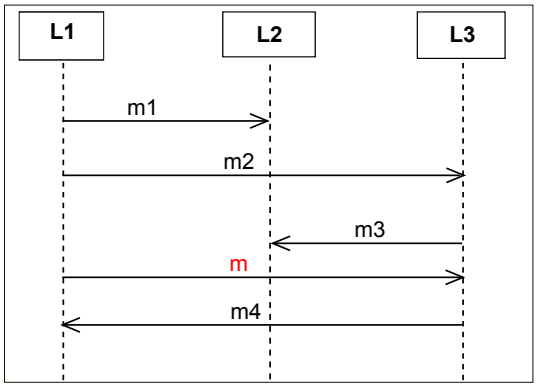

FIGURE 7.14 – Diagramme de séquence abstrait

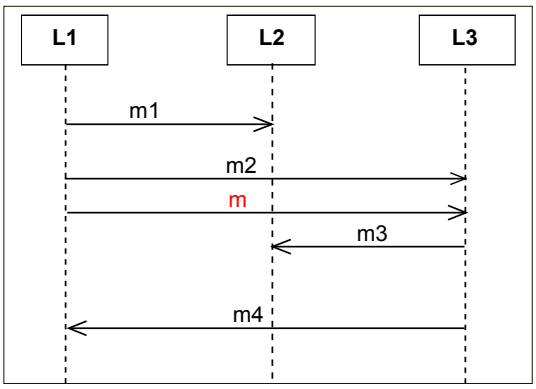

FIGURE 7.15 – Diagramme de séquence raffiné du DS de la Figure 7.14

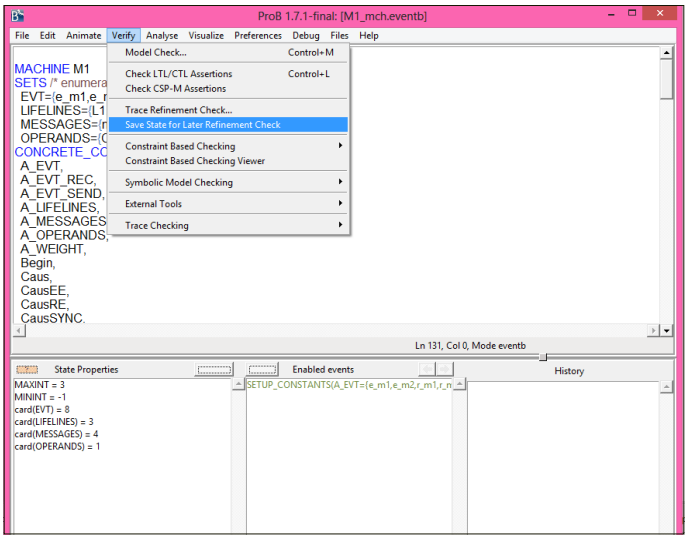

FIGURE 7.16 – Sauvegarde des états pour la vérification du raffinement

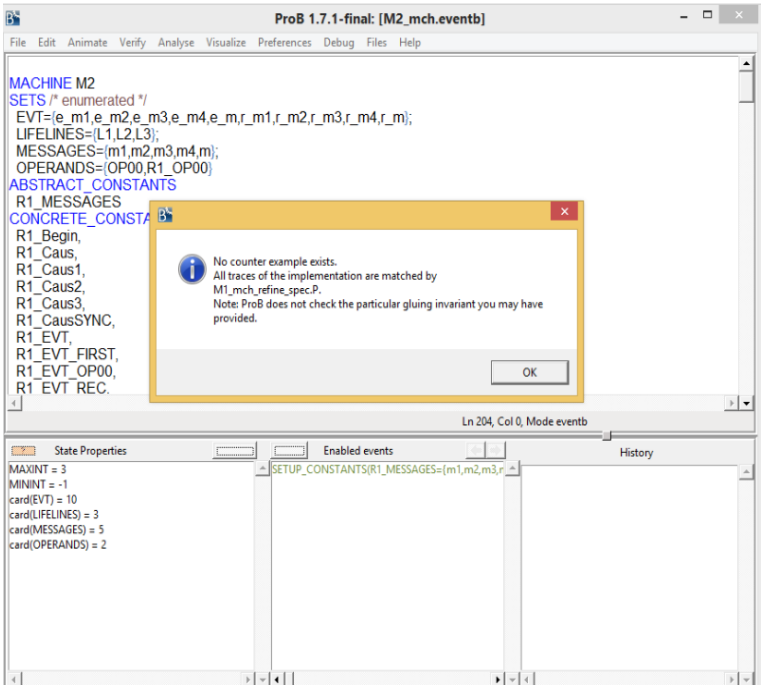

FIGURE 7.17 – Raffinement de traces entre les machines des DS encodés des Figures 7.14 et 7.15

• Nous considérons le cas de de raffinement d'un DS basique en introduisant de nouveaux événements. Soit le diagramme de séquence basique représenté par la Figure 7.18 contenant deux lignes de vie $L1$ et $L2$ qui échangent deux messages $m1$ et $m2$. Nous proposons le raffinement de ce diagramme (Figure 7.19) en substituant la ligne de vie $L2$ par les lignes de vie $L2.1$ et $L2.2$. Deux nouveaux messages $m1.1$ et$m1.2$ sont rajoutés et échangés entre les nouvelles lignes de vie.

ProB nous permet de prouver qu'il existe une relation de simulation entre la spécification concrète et la spécification abstraite par conséquent nous affirmons que la spécification concrète est un raffinement de la spécification abstraite. Cependant nous avons un échec de vérification

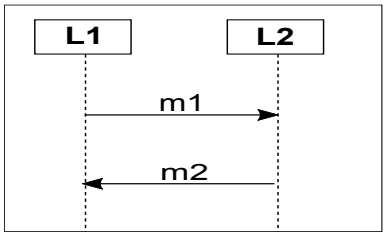

FIGURE 7.18 – DS1 : Diagramme de séquence basique

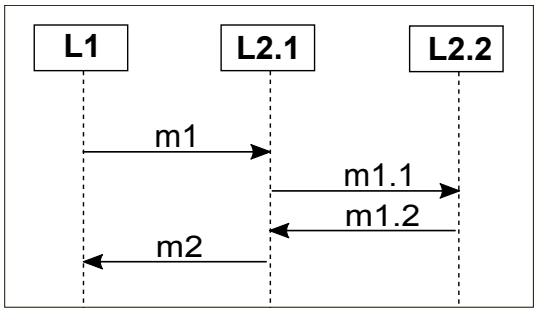

FIGURE 7.19 – DS2 : raffinement du diagramme de séquence de la Figure 7.18

de la relation de raffinement de traces. Les Figures 7.20 et 7.21 représentent respectivement le diagramme état transition de la machine abstraite et concrète des DS encodés.

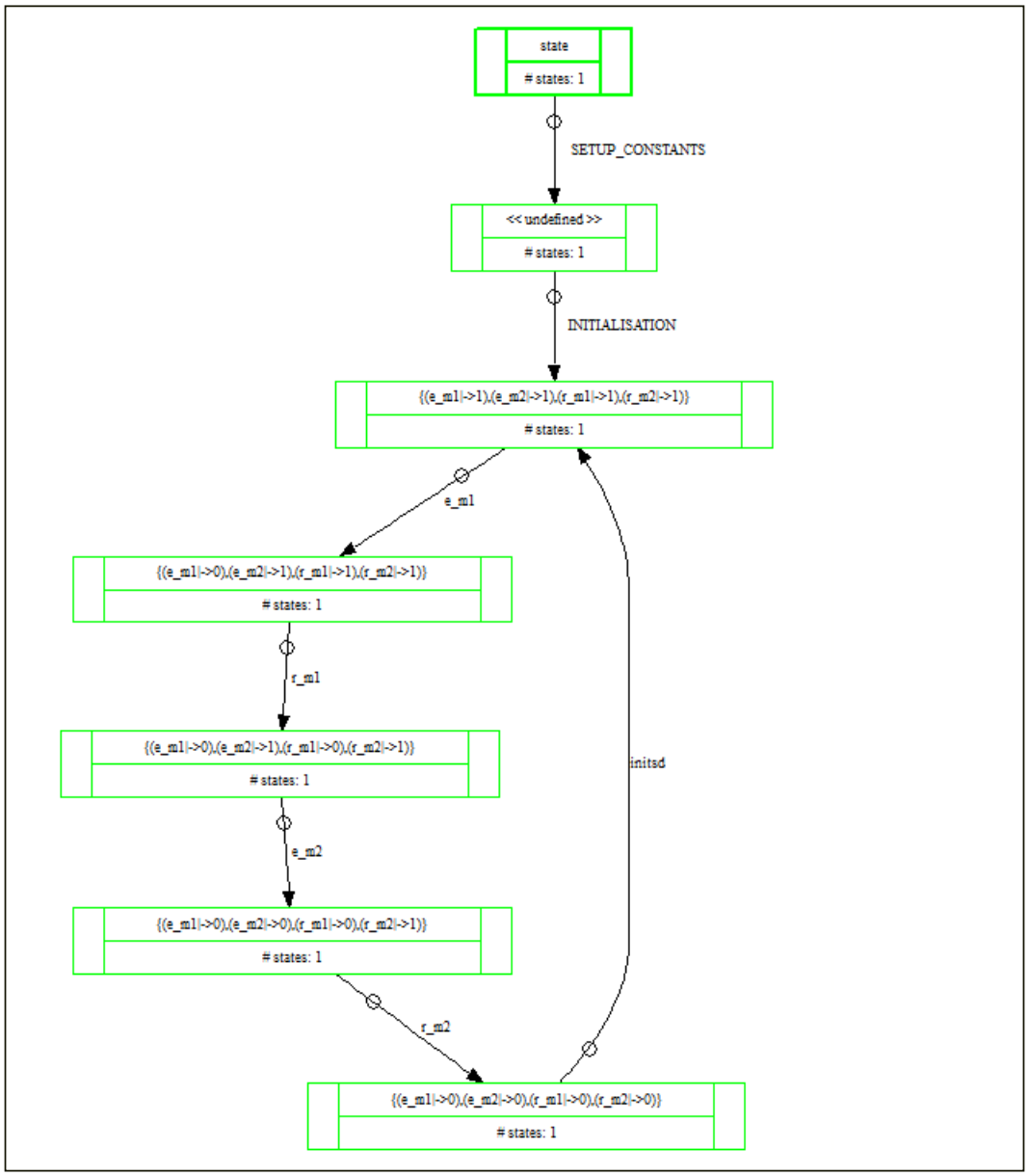

FIGURE 7.20 – Diagramme état transition de la machine abstraite du DS de la Figure 7.18

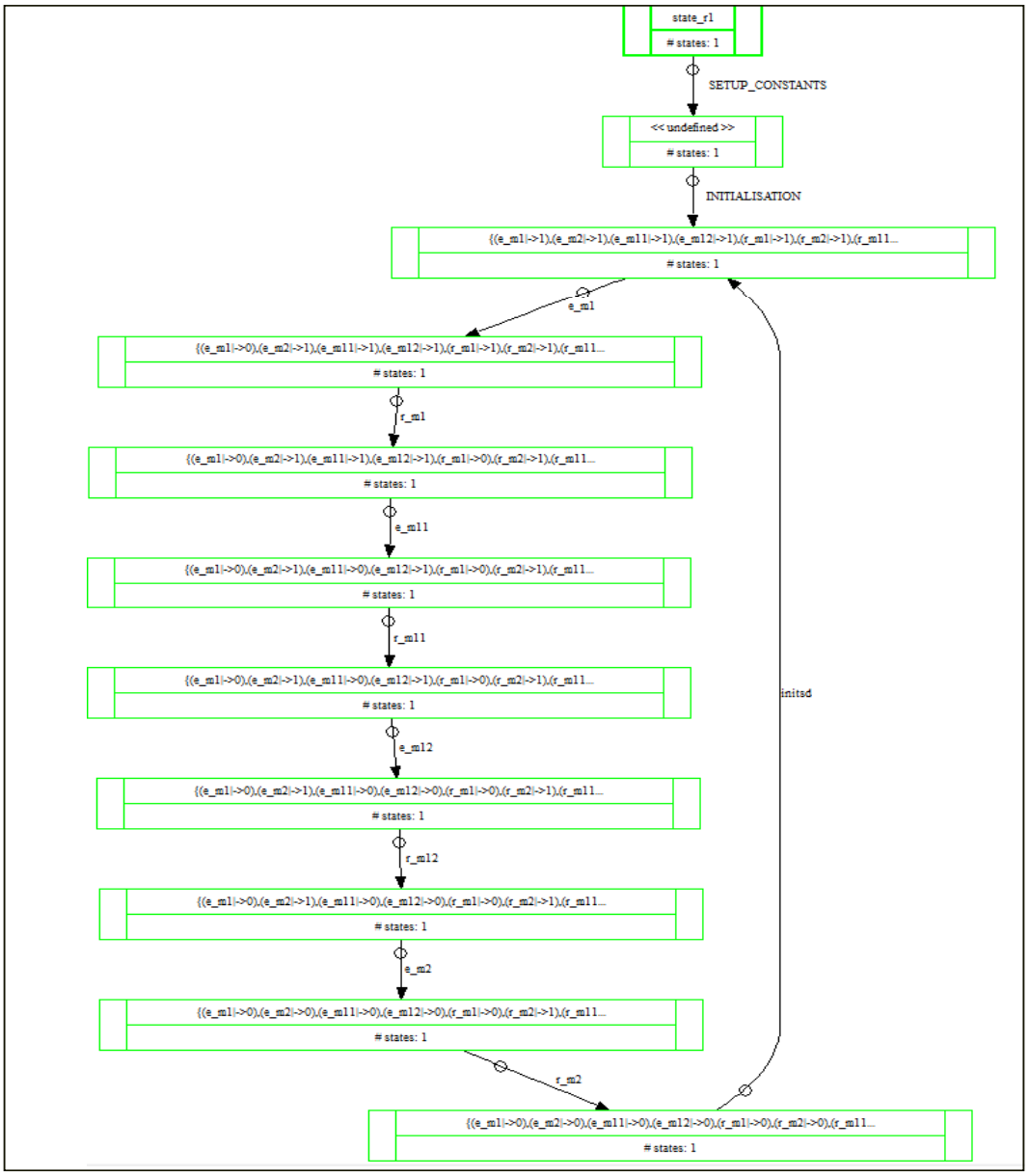

FIGURE 7.21 – Diagramme état-transition de la machine concrète du DS de la Figure 7.19

Pour conclure, à chaque fois où nous avons des nouveaux événements, ProB ne prouve pas qu'il y a un raffinement de traces entre les machines relatives aux DS considérés en effet, l'explorateur de modèle ne possède pas des mécanismes pour cacher les nouveaux événements. Cependant nous parvenons à vérifier que le comportement de la machine raffiné simule le comportement de la machine abstraite.

7.4.2 Cas de raffinement d'un diagramme de séquence comprenant des FC imbriqués

7.4.2.1 Discussion sur le raffinement des fragments combinés

Dans le Chapitre raffinement 5, nous avons étudié théoriquement le raffinement des fragments combinés. Nous avons dit que certains FC peuvent être raffinés en les substituant par d'autres FC. Rappelons les exemples considérés :

• le DS abstrait contient deux fragments combinés OPT est raffiné par un DS contenant un ALT avec deux opérandes (exemple1 : Figure 5.10, Figure 5.11),

• le DS abstrait contient un fragment combiné LOOP avec N itérations, est raffiné par un DS contenant un fragment combiné LOOP avec un nombre d'itérations qui est inférieur à N (exemple2 : Figure 5.7, Figure 5.8),
• le DS abstrait contient un fragment combiné LOOP avec N itérations, est raffiné par un DS contenant deux fragments combinés LOOP imbriqués dont la multiplication des nombres d'itérations de chaque LOOP est égale à N (exemple3 : Figure 5.7, Figure 5.9).
Dans tous ces exemples, les DS possèdent le même alphabet et le DS concret génère moins de traces que le DS abstrait. En encodant ces exemples vers des spécifications en B événementiel et en les testant avec l'explorateur des modèles ProB nous avons eu aussi bien un échec pour prouver qu'il existe une relation de simulation et un échec pour prouver qu'il existe une relation de raffinement de traces entre les machines abstraites et concrètes des DS encodés de chaque exemple. Ceci s'explique puisque dans notre approche nous considérons l'évaluation de la garde et la synchronisation entre les lignes de vie qui se fait avec des événements fictifs qui sont supposés être des événements internes (mais puisqu'ils agissent sur les états des autres événements cette hypothèse est rejetée). Le nombre d'événements fictifs de chaque DS abstrait des exemples considérés et le nombre d'événements fictifs de chaque DS abstrait n'est pas forcément le même (exemple 1 et 3). D'autant plus que l'évaluation de la garde dans le DS abstrait et dans le DS concret ne se fait pas de la même manière. Pour toutes ces raisons dans le DS concret, nous aurons des états qui n'ont pas leurs contreparties dans le DS concret. Par conséquent les résultats fournis par ProB s'expliquent bien.

7.4.2.2 Raffinement des interactions dans une application web

Considérons une application web ayant des composants distribués (Figure 7.22) : un client, un serveur web et une application. Ces composants sont modélisés par les lignes de vie *Client*, *WebServer* et *Application*. Initialement, le client se connecte au serveur puis il peut formuler au maximum deux requêtes. Le serveur lance la recherche en interagissant avec une application. Deux cas peuvent être envisagés : soit la requête est trouvée, dans ce cas le résultat est transmit au client, soit la requête est non trouvée, dans ce cas le serveur affiche un message indiquant ce résultat. Le serveur peut effectuer au maximum 3 recherches pour chaque requête.

✓ Modélisation par les diagrammes de séquence

• **Premier raffinement.** Dans un premier raffinement (Figure 7.23), nous proposons le raffinement du composant serveur qui est modélisé par le composant *WebServer* dont nous détaillons son comportement interne en rajoutant le composant système d'authentification qui est modélisé par la ligne de vie *AuthSys*. Le serveur échange deux nouveaux messages avec le système d'authentification qui sont : $authentificate$ et *accept*.

• **Second raffinement.** Dans un deuxième raffinement (Figure 7.24), nous raffinons le composant application qui est modélisé par la ligne de vie *Application* par les deux composants serveur d'application et base de données du serveur qui sont respectivement modélisés par les deux lignes de vie *ApplicationServer* et *DBServer*. Les événements abstraits *r_seek_query* et *e_result_query* de la ligne de vie abstraite *Application* sont distribués dans le diagramme de séquence raffiné entre les nouvelles lignes de vie *ApplicationServer* et *DBServer*, qui substituent la ligne de vie abstraite. Les deux nouvelles lignes de vie échangent un nouveau message *db_query*.

• **Troisième raffinement.** Dans ce raffinement (Figure 7.25), nous proposons le raffinement du message d'authentification $authentificate$ en le décomposant en deux messages demande d'authentification $authentificate\_request$ et la soumission des informations de l'authentification *submit_credentials*. Les deux messages sont émis par la même ligne de vie que celle du message abstrait raffiné, qui est serveur web *WebServer*, et ils sont également reçus par la

même ligne de vie de l'événement abstrait raffiné, qui est le système d'authentification $AuthSys$.

✓ Bref description des modèles B événementiel des diagrammes de séquence encodés :

• Le modèle en B événementiel relatif au diagramme de séquence abstrait de la Figure 7.22 est composé des contextes $ctx1$, $ctx1P$ et la machine $M1$. La Figure 7.27 représente le modèle fourni par ProB. Dans le contexte $ctx1$, nous exprimons des axiomes qui représentent des propriétés sur les constantes et les ensembles du DS abstrait encodé. Dans le contexte $ctx1P$, qui étend (EXTENDS) le contexte $ctx1$ nous donnons le paramétrage des constantes et des ensembles. Dans la machine abstraite $M1$ qui a une visibilité (SEES) sur le contexte $ctx1$, nous définissons les variables du modèle qui sont constituées de la variable *state* (qui exprime l'état de chaque événement), la variable $current\_lifeline$ dans laquelle nous spécifions la ligne de vie courante de l'événement en cours et les variables gi qui représentent les gardes des opérandes du DS abstrait. Dans la clause (INVARIANT), nous fournissons le typage des variables. La partie dynamique de la machine est constituée de la clause INITIALISATION dans laquelle nous initialisons les variables du modèle : la variable *state* est initialisée par les poids des événements, la variable *begin* est initialisée par le premier événement qui est autorisé à se déclencher, la variable $current\_lifeline$ qui est initialisée par la ligne de vie du premier événement qui est autorisé à se déclencher et les variables gi dont chacune est initialisée par une valeur de l'ensemble des booléens.
Les événements de la machine M1 sont composés par les événements du DS abstraits et les événements fictifs qui sont associés aux opérandes du DS abstrait et qui sont requis pour l' évaluation de la garde et la synchronisation des lignes de vie pour chaque opérande.

• Le modèle en B événementiel relatif au diagramme de séquence raffiné (Figure 7.23) est composé des contextes $ctx2$, $ctx2P$ et la machine $M2$. La Figure 7.29 représente le modèle fourni par ProB. Dans le contexte $ctx2P$, qui étend (EXTENDS) le contexte $ctx2$ nous donnons le paramétrage des constantes et des ensembles relatifs au premier DS raffiné encodé de la Figure 7.23.
Dans la machine abstraite $M2$ qui a une visibilité (SEES) sur le contexte $ctx2$, nous définissons les variables du modèle qui sont les mêmes que dans le modèle abstrait sauf que pour les variables qui indiquent respectivement l'état de chaque événement et la ligne de vie courante, nous rajoutons le suffixe $\_r1$ soit $state\_r1$ et $current\_lifeline\_r1$. Dans la clause invariant, en plus des invariants de typage des variables, nous rajoutons l'invariant de collage qui permet de lier les variables concrètes avec les variables abstraites comme suit.

$$New1\_EVT \lhd state\_r1 = state$$

Dans le DS raffiné nous avons introduit de nouveaux événements par conséquent nous définissons dans la clause VARIANT une expression arithmétique strictement positive qui est décrémentée à chaque fois où un nouveau événement est déclenché.
Les événements de la machine raffinée sont composés des événements du DS raffiné considéré, des événements fictifs et des nouveaux événements (qui ont un statut convergent).
Dans ce raffinement, la ligne de vie serveur est substituée par les deux lignes de vie serveur web et système d'authentification ; ces nouvelles lignes de vie échangent de nouveaux messages d'authentification et d'acceptation. Dans les contextes $ctxMap$ et $ctxMapP$ nous exprimons les axiomes qui permettent d'assurer la correspondance entre les éléments abstraits et raffinés ainsi que les règles de raffinement à respecter.
Par exemple l'axiome $Map\_LIFELINES \in R1\_LIFELINES \longrightarrow A\_LIFELINES$: permet de faire la correspondance entre les lignes de vie concrètes et abstraites ;
l'axiome $A\_FCT\_LIFELINES = ((A\_EVT \lhd A\_FCT\_LIFELINES); Map\_LIFELINES)$ permet d'exprimer que les lignes de vie abstraites doivent garder leurs événements abstraits et les nouvelles lignes de vie héritent des événements abstraits de la ligne de vie substituée dans le

DS raffiné.
N.B. dans les axiomes, nous désignons par le préfixe A : abstrait et R1 : premier raffinement.
Dans la même logique nous définissons les modèle B pour les autres DS raffinés encodés.

• Le modèle en B événementiel relatif au diagramme de séquence raffiné (Figure 7.24) est composé des contextes $ctx3$, $ctx3P$ et la machine $M3$. La Figure 7.31 représente le modèle fourni par ProB.

Dans ce raffinement, le comportement interne de la ligne de vie serveur d'application est détaillé en lui rajoutant la ligne de vie base de données serveur. Un nouveau message, requête base de données, est échangé entre ces lignes de vie.

Nous exprimons les correspondances entre les éléments abstraits et raffinés des machines $M2$ et $M3$ dans les axiomes des contextes $ctxMap2$ et $ctxMapP2$. Dans la machine M3 nous exprimons dans la clause INVARIANT, en plus du typage des variables, l'invariant de collage :
$New2\_EVT \lhd state\_r2 = state\_r1$
Nous exprimons également dans la clause VARIANT l'expression mathématique suivante :
$(state\_r2(e\_db\_query) + state\_r2(r\_db\_query))$ qui garantit la non divergence du nouveau événement introduit db_query.

• Le modèle en B événementiel relatif au diagramme de séquence raffiné (Figure 7.25) est composé des contextes $ctx4$, $ctx4P$ et la machine $M4$. La Figure 7.33 représente le modèle fourni par ProB.

Dans ce raffinement le message d'authentification *authenticate* est décomposé par deux messages qui sont liés par la relation de précédence $_{EE}$: le message *authentication_request* ensuite le message *submit_credentials*. Cette décomposition va se répercuter sur les événements, *e_authenticate* et *r_authenticate*, associés aux messages décomposés. En effet l'événement *e_authenticate* est décomposé en deux événements *e_submit_credentials* et *e_authentication_request*.
Ces deux derniers événements sont de différents types :
i) l'événement *e_authentication_request* raffine *skip* (c'est un nouveau événement),
ii) l'événement *e_submit_credentials* raffine l'événement abstrait *e_authenticate*.

L'événement *r_authenticate* est décomposé en trois événements *r_authentication_request*, *r_submit_credentials* et *r_authenticate*. Les événements correspondent à la réception des deux nouveaux messages et l'événement *r_authenticate* est ajouté pour assurer que les nouveaux événements ont bien été reçus. Ces événements sont de deux types :
i) les événements *r_authentication_request* et *r_submit_credentials* raffinent *skip*. L'action de ces deux événements mènent à des nouveaux états qui ne sont pas liés aux état abstraits.
ii) l'événement *r_authenticate* raffine l'événement abstrait *r_authenticate*.

Dans les contextes $ctxMap3$ et $ctxMapP3$, nous définissons quelques axiomes dans lesquels nous exprimons la correspondance entre les éléments des machines $M3$ et $M4$.
Dans la machine M4 nous exprimons dans la clause INVARIANT en plus du typage des variables, les invariants de collage suivants :

$$New3\_EVT \lhd state\_r3 = e\_authenticate \lhd state\_r2$$

$$state\_r3(e\_submit\_credentials) = state\_r2(e\_authenticate)$$

Nous exprimons également dans la clause VARIANT l'expression mathématique suivante :
$(state\_r3(e\_authentication\_request)+state\_r3(r\_authentication\_request)+state\_r3(r\_submit\_credentials))$
qui garantit la non divergence des nouveaux événements introduits.

✓Analyses des spécifications avec ProB :
l'analyse par ProB prouve que nous avons une relation de simulation entre les machines raffinés,

par conséquent il existe une relation de raffinement entre les DS encodés. Les Figures 7.26, 7.28, 7.30 et 7.32 représentent respectivement les interfaces ProB des machines témoignant de leurs analyses par l'explorateur des modèles. Nous récapitulons quelques informations pertinentes de cette analyse dans le Tableau 7.1.
Les Figures 7.27, 7.29, 7.31 et 7.33 représentent respectivement les architectures relatives à la machine abstraite et ses machines raffinées. Ces architectures sont générées automatiquement par ProB et elles sont conformes au modèle de traduction générique pour la machine abstraite et au modèle générique étendu pour la vérification de la relation de raffinement que nous avons proposé.

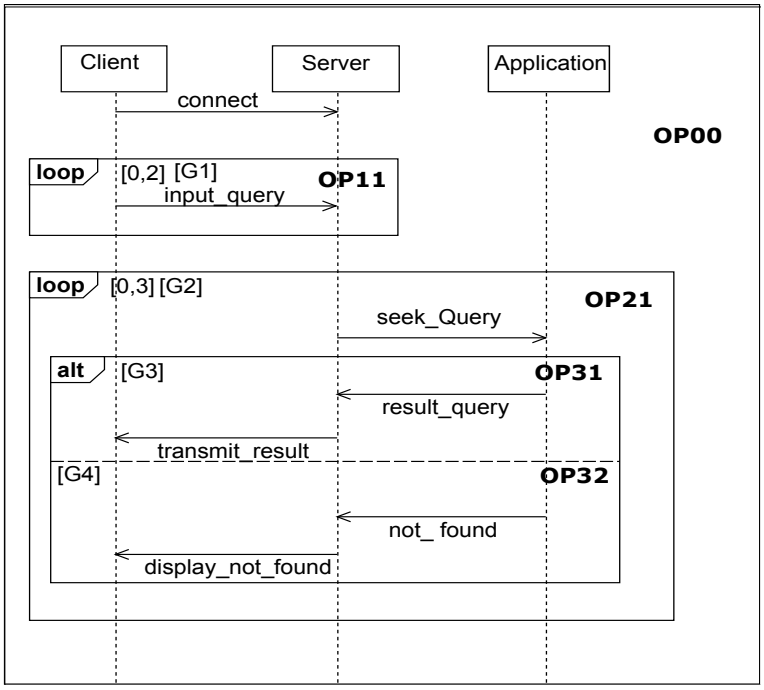

FIGURE 7.22 – Interactions avec une application web

Machine	Nombre d'états	Nombre de transitions	Temps d'exécution
M1	3330	6625	14736044 microsecondes/itération
M2	4930	10257	32278670 microsecondes/itération
M3	5506	11217	45186462 microsecondes/itération
M4	8130	18033	153487046 microsecondes/itération

TABLE 7.1 – Illustration des analyses des machines fournies par ProB

7.5 Conclusion

En se basant sur le modèle générique de traduction des diagrammes de séquence d'UML2.X vers des spécifications en B événementiel, nous avons étendu ce modèle pour aboutir à un modèle générique de vérification de la relation de raffinement entre les diagrammes de séquence. Nous avons mené plusieurs expérimentations à travers l'étude de plusieurs exemples en exploitant les outils de la méthode B événementiel (le prouveur de théorèmes (Rodin) et l'explorateur de modèles (ProB)). Ce travail a fait l'objet d'une publication dans une conférence internationale [Fat16].

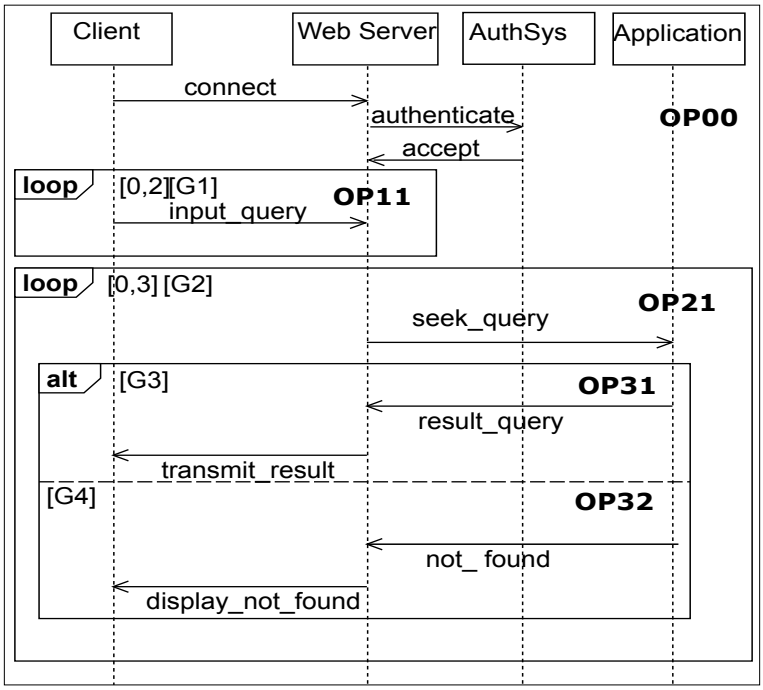

FIGURE 7.23 – Premier raffinement du diagramme de séquence de la Figure 7.22

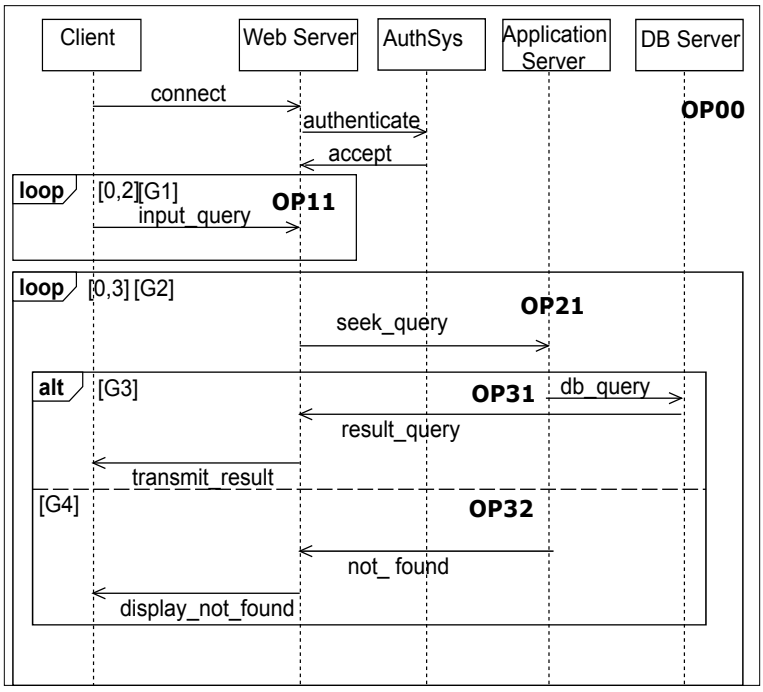

FIGURE 7.24 – Deuxième raffinement du diagramme de séquence de la Figure 7.23

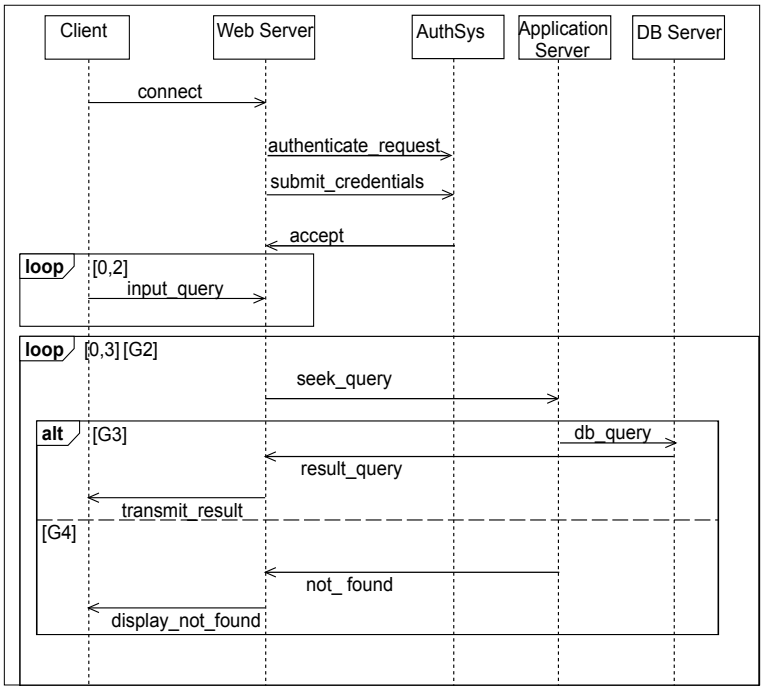

FIGURE 7.25 – Troisième raffinement du diagramme de séquence de la Figure 7.24

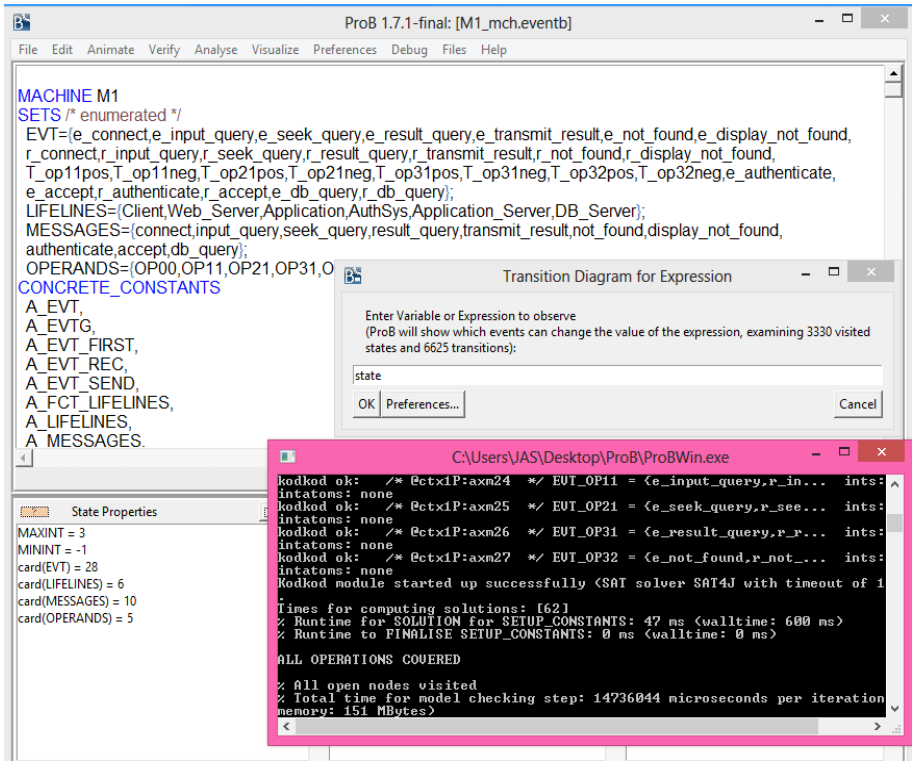

FIGURE 7.26 – Interface ProB de la première machine

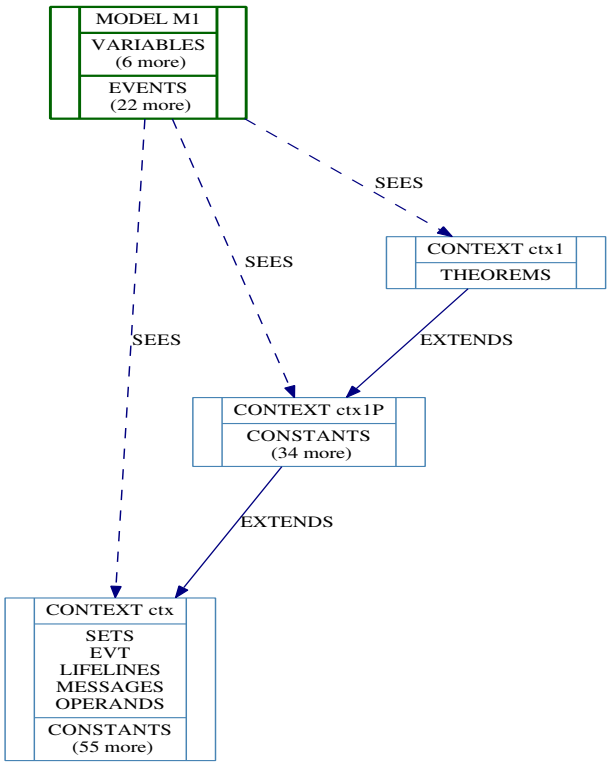

FIGURE 7.27 – Architecture du modèle B du DS abstrait encodé

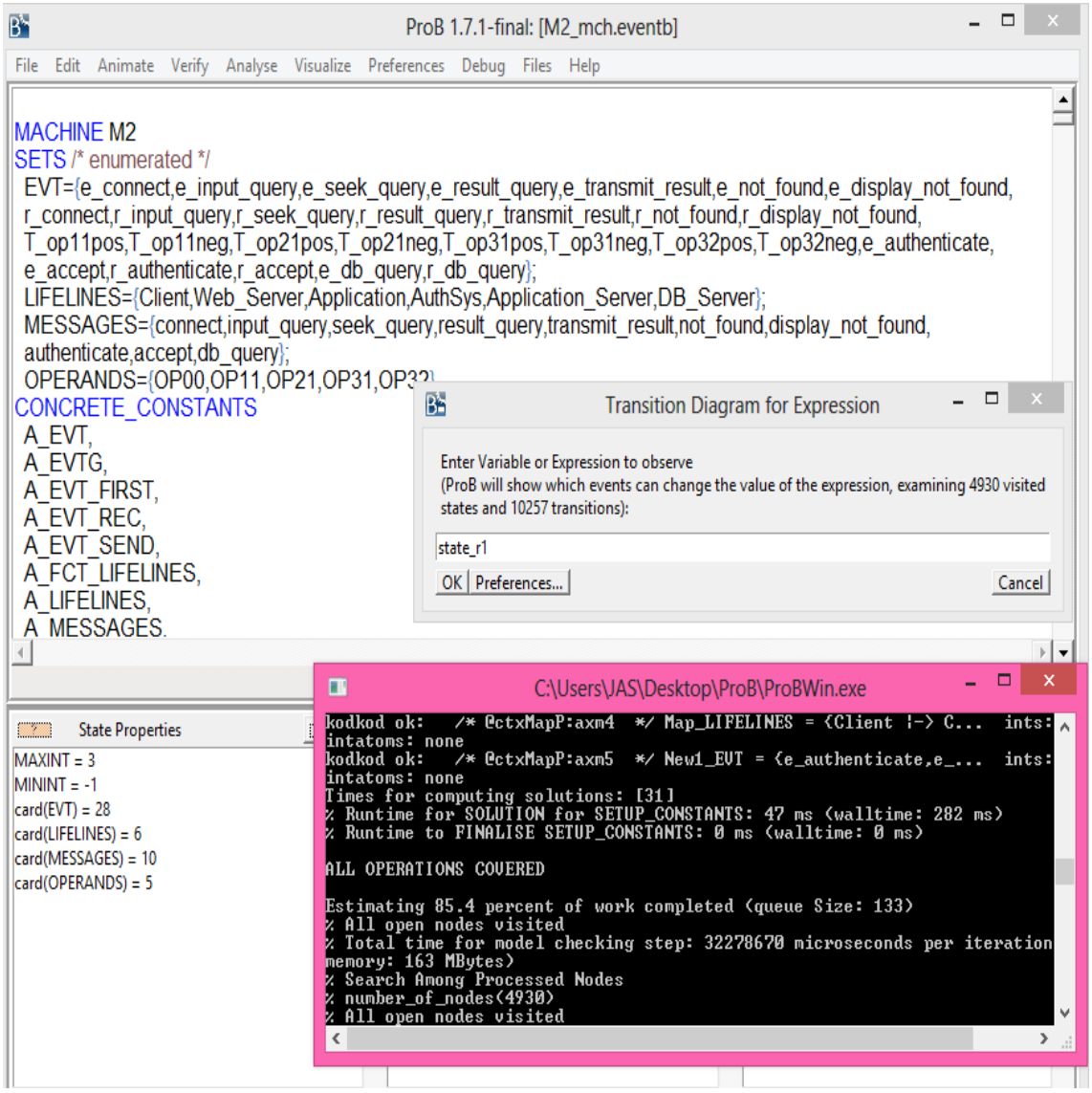

FIGURE 7.28 – Interface ProB de la première machine raffinée

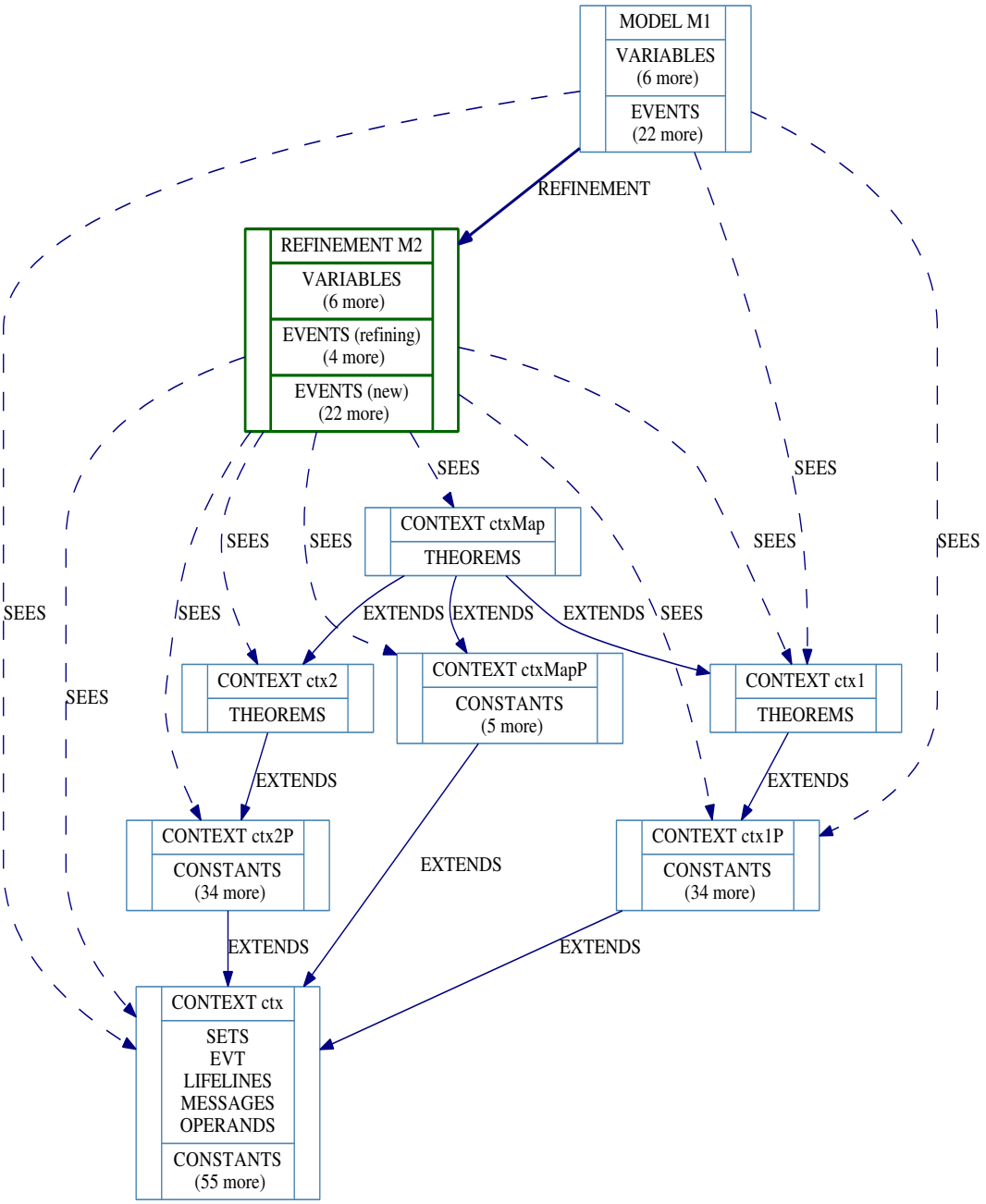

FIGURE 7.29 – Architecture du modèle B du premier DS raffiné encodé

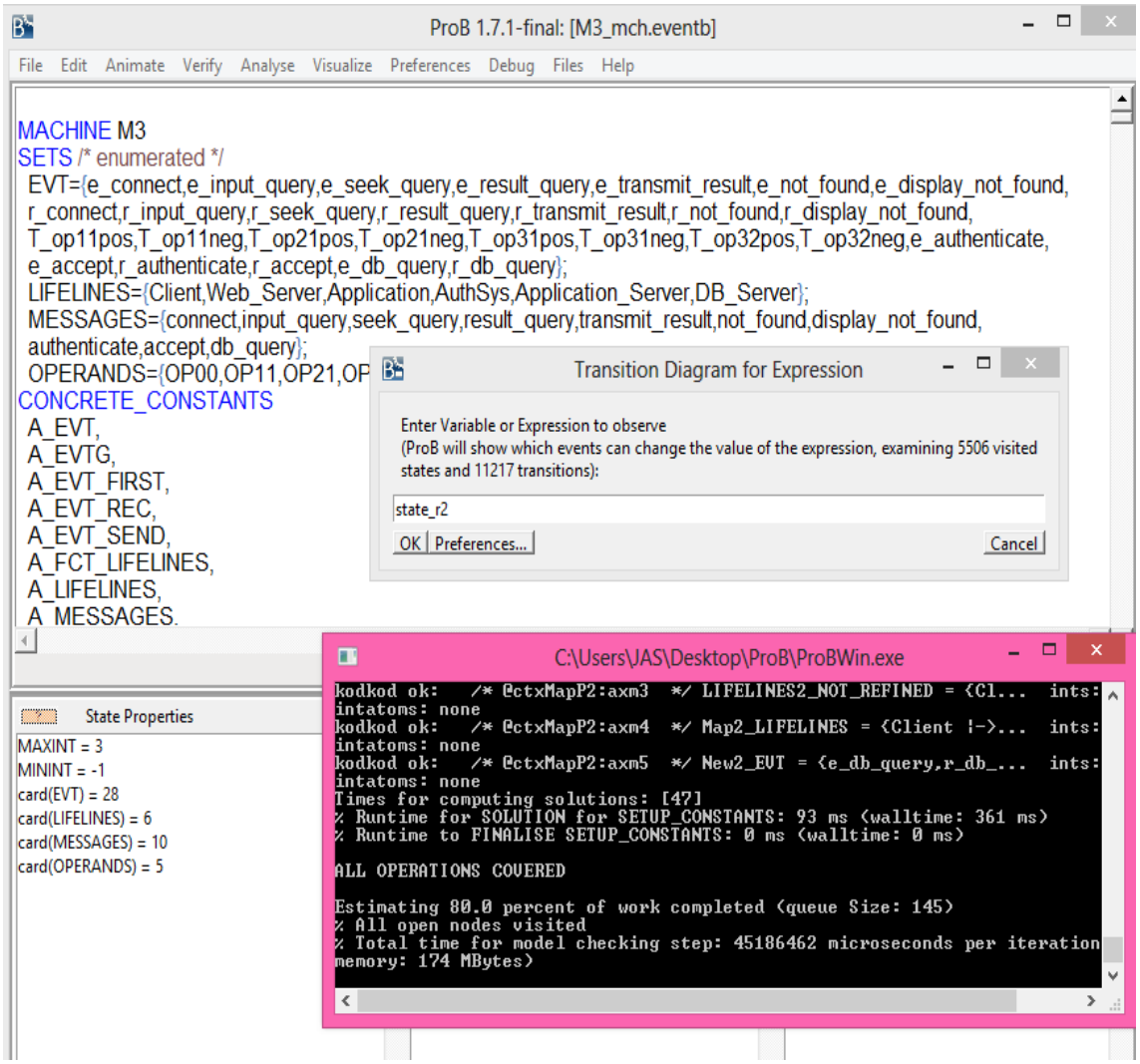

FIGURE 7.30 – Interface ProB de la deuxième machine raffinée

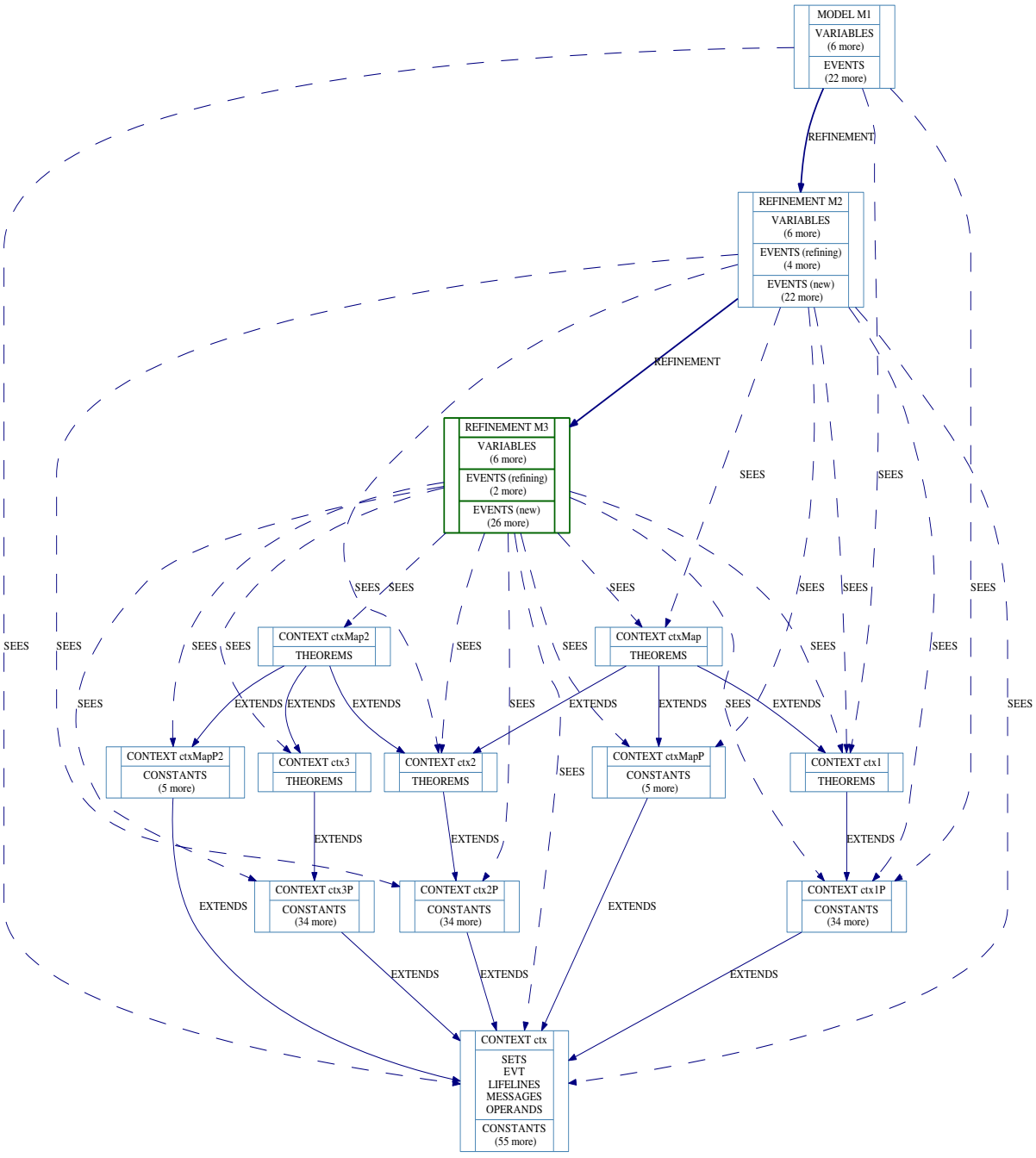

FIGURE 7.31 – Architecture du modèle B du deuxième DS raffiné encodé

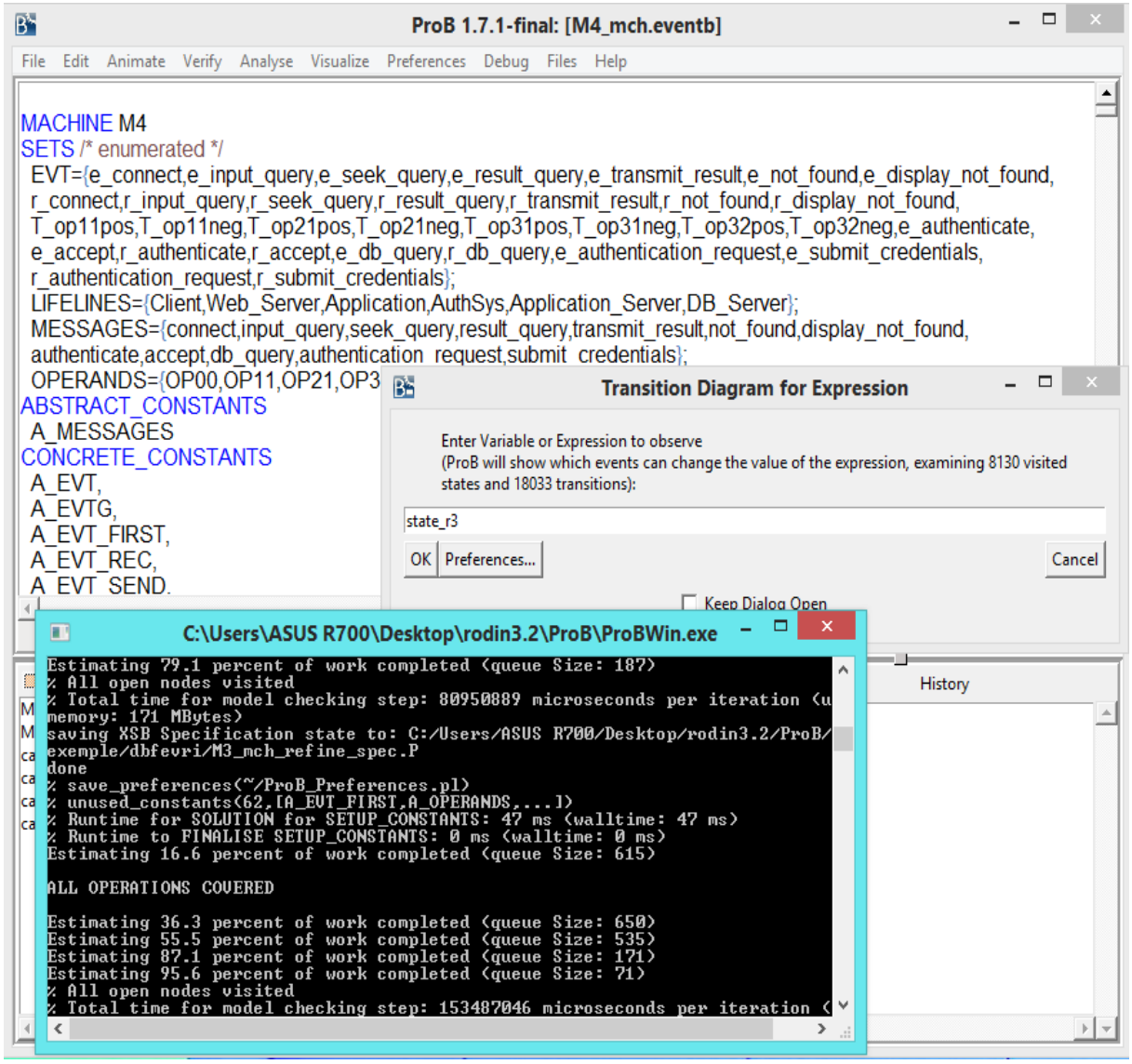

FIGURE 7.32 – Interface ProB de la troisième machine raffinée

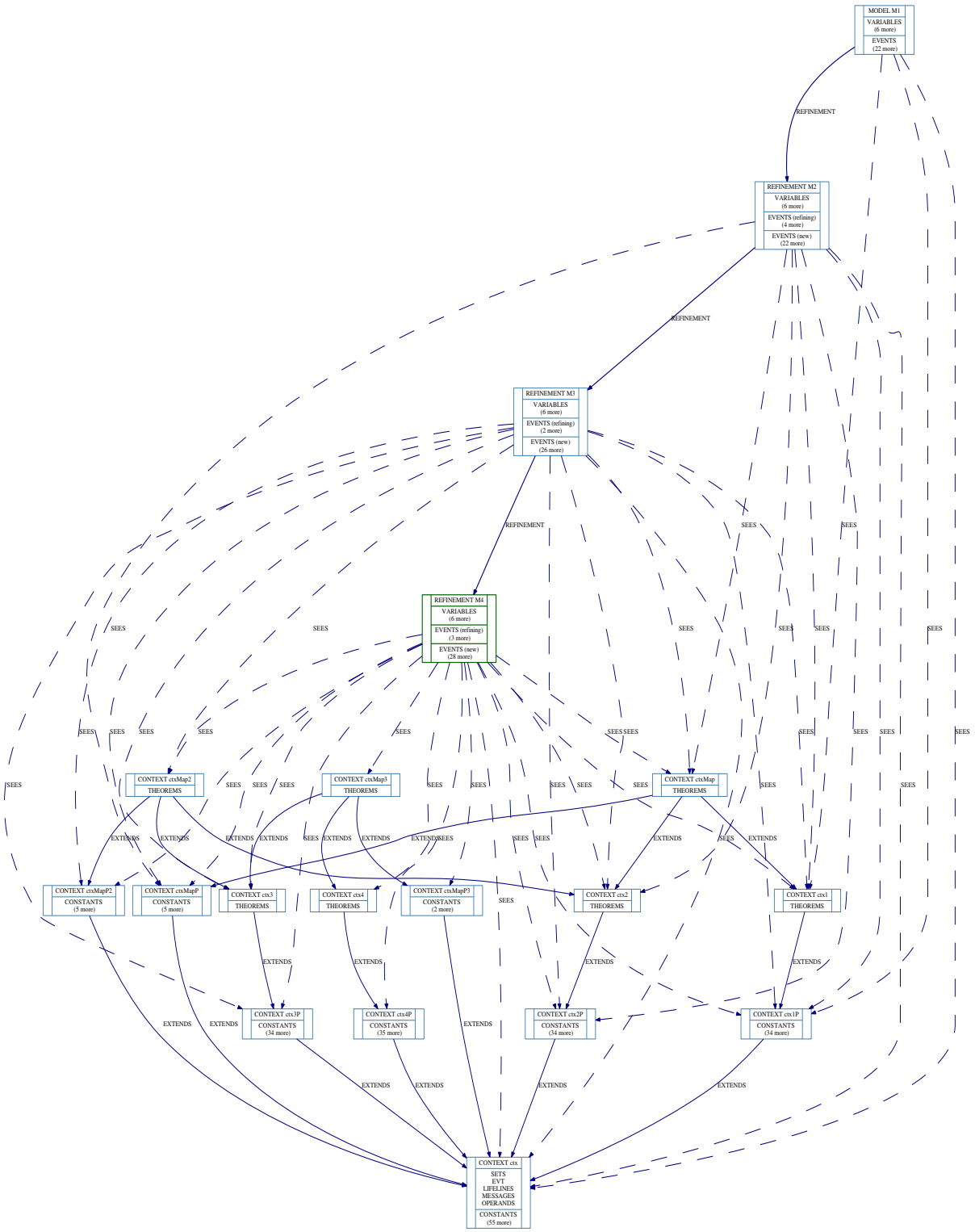

FIGURE 7.33 – Architecture du modèle B de la troisième DS raffiné encodé

Chapitre 8

Conclusion générale et perspectives

Ce travail a fait l'objet de la proposition d'un cadre théorique et pragmatique pour le développement incrémental de diagrammes de séquence d'UML2.X modélisant les comportements des systèmes distribués. Bien que cette approche de construction incrémentale ait largement déployée pour d'autres formalismes tels que les automates, les systèmes de transitions, voire les *statecharts*, cependant pour UML peu de méthodes d'aide à la construction incrémentale ont été proposées et peu d'outils offrent des moyens d'évaluation des diagrammes utilisés.

La construction incrémentale consiste à effectuer une suite de transformations sur un modèle abstrait comprenant le minimum de composants et modélisant des fonctionnalités réduites du système. Les transformations consistent à l'ajout ou la suppression de composants ou des interactions jusqu'à l'obtention d'un modèle ayant des détails suffisants pour qu'il soit implémenté. Chaque modèle transformé doit être cohérent avec le modèle antérieur.

La vérification des modèles obtenus par raffinements successifs nécessite l'association d'une sémantique formelle. Les sémantiques existantes des diagrammes de séquence sont de divers types (dénotationnelle, transformationnelle, opérationnelle, etc...), dont chaque type de sémantique a ses avantages et ses limites.

L'apport de notre approche est double puisque nous avons d'abord fourni aux diagrammes de séquence d'UML2.X une sémantique opérationnelle qui est indépendante de tout formalisme cible, puis nous avons fourni une sémantique transformationnelle en les traduisant vers des spécifications en B événementiel.

En effet, les relations de raffinement sont bien établies sur les systèmes de transitions étiquetées (LTS). Ce qui justifie notre choix pour la définition d'une sémantique opérationnelle qui est concrètement donnée comme un LTS. Dans la sémantique opérationnelle nous avons explicité de façon non ambiguë la génération du comportement d'un DS et nous avons spécifié l'occurrence de tous les types des événements dans un diagramme de séquence avec des fragments combinés imbriqués. Un facteur déterminant pour l'occurrence de chaque événement est la détermination de ses relations de précédence. Or la détermination des relations de précédence n'est pas aussi évidente pour un DS comportant des FC imbriqués.

Cette difficulté explique les hypothèses restrictives (telles que les interprétations non standard de certains FC, aplatissement des diagrammes de séquence avec fragments combinés, etc...) dans les approches existantes pour détourner les problèmes et les inconsistances pour le calcul des relations de précédence. En outre les définitions de la sémantique standard sont informelles et ambiguës. Par conséquent, pour remédier aux insuffisances de la sémantique standard ainsi que celles des sémantiques existantes qui ont été proposées pour les diagrammes de séquence d'UML2.X, nous avons proposé une sémantique ayant des relations bien définies et formalisées qui permettent de déterminer l'ordre partiel entre les événements d'un DS modélisant le comportement des systèmes distribués.

La définition d'une sémantique opérationnelle qui est basée sur une sémantique de calcul de l'ordre partiel et qui est adéquate aux DS modélisant les comportements des systèmes distribués fut une base solide pour l'implémentation de l'approche. Nous avons explicité d'abord de façon informelle les règles de raffinement des diagrammes de séquence d'UML2.X, puis nous avons formalisé les relations de correspondances qui doivent être vérifiées pour prouver qu'il y a une relation de raffinement entre les diagrammes de séquence considérés.
Pour bénéficier de la multitude d'avantages de la méthode formelle B événementiel qui, outre sa sémantique bien définie, offre des outils puissants qui permettent la vérification de la cohérence de notre sémantique, l'expression de la vérification de quelques propriétés des systèmes spécifiés et la vérification de la relation de raffinement entre les sémantiques opérationnelles relatives aux DS considérés. Nous avons défini un modèle générique de traduction des DS avec FC imbriqués. Sur la base de ce modèle, nous l'avons étendu pour aboutir à un modèle de vérification de la relation de raffinement en B événementiel.

Perspectives

• La sémantique causale peut être étendue pour inclure d'autres aspects des diagrammes de séquence que nous n'avons pas considérés tels que les aspects temporels et les portes (*gate*) (Figure 8.1). Les *gates* constituent des éléments syntaxiques qui peuvent parfois causer un problème de mauvaise formation (*ill-formedness*) des diagrammes de séquence, étant donné que nous associons à chaque message un événement émit et un événement recu. Le problème est très visible dans le cas où un *gate* traverse un fragment combiné LOOP comme illustré dans la Figure 8.1 où nous avons une seule émission du message $m1$ et trois réceptions du même message.
• Dans notre approche nous nous sommes intéressés aux comportements valides des systèmes étudiés. Une extension possible sera la prise en compte du fragment combiné NEG qui permet de modéliser des comportements invalides, ainsi que les autres FC qui n'ont pas été considérés (ASSERT, CONSIDER, IGNORE, etc...).
• Même si pour traiter certains aspects, tels que le problème de l'évaluation des gardes dans les FC, nous avons émis une hypothèse restrictive pour éviter un problème connu dans la littérature qui est le choix non local (*non-local choice*) [J. 05] (Figure 8.2), notre approche résout partiellement ce problème avec l'utilisation des événements fictifs. Cependant cette solution peut être améliorée pour apporter une solution complète à ce problème. Le choix non local se pose lorsque nous avons plusieurs premiers événements (*first events*) qui sont autorisés à se déclencher. Dans un DS basique nous pouvons avoir plusieurs premiers événements. Cependant ceci ne pose pas un problème puisque notre sémantique est une sémantique de entrelacement (*interleaving semantics*) qui garantie qu'à un instant donné, nous avons au plus un événement qui se déclenche. Le problème se pose réellement lorsque nous un FC ALT, nous avons plusieurs premiers événements qui sont autorisés à se déclencher dans un même opérande. La Figure 8.2 illustre ce cas : dans l'opérande $OP12$ nous avons deux premiers événements ($!m2$ et $!m3$) qui sont autorisés à se déclencher.

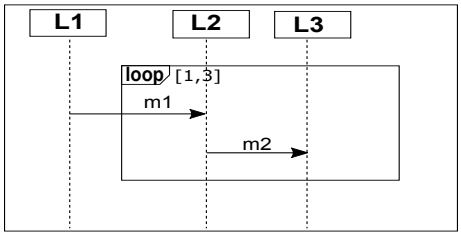

FIGURE 8.1 – Illustration de *gate* dans un FC LOOP

• La proposition d'un *plugin* sur Rodin qui permet d'automatiser le calcul des relations de précédence que nous avons défini et formalisé dans la sémantique causale est aussi une perspective intéressante pour ce travail.

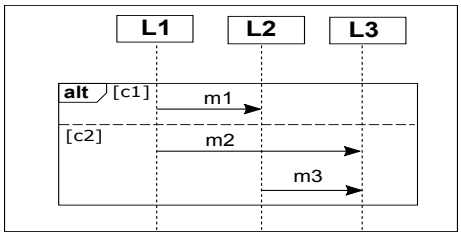

FIGURE 8.2 – Illustration du problème du *choix non local* dans un FC ALT

• Pour la formalisation de la relation de raffinement, nous avons eu recours aux relations de raffinement qui sont définies sur les systèmes de transitions étiquetées et nous avons émis quelques hypothèses pour la préservation des comportements requis. Les travaux récents de [D. 09] ont considéré le formalisme des systèmes de transitions modaux (*modal transition system (MTS)*), qui sont une extension de systèmes de transitions étiquetées, et qui supportent, dans leurs définitions, les comportements requis et possibles. Ils ont formalisé la relation de raffinement raffinement de branchement **branching refinement** qui mixe les avantages des relations de raffinement faible (*weak refinement*) et fort (*strong refinement*).
Cependant, le formalisme MTS considéré ignore la condition de garde. Une perspective possible est d'étendre les MTS afin d'inclure la garde et vérifier la possibilité de formaliser le raffinement de branchement dans la méthode B événementiel.

Annexe A

La méthode B événementiel

Dans cette annexe, nous donnons un aperçu sur la méthode formelle B événementiel ainsi que le langage associé. En effet, B événementiel est basée sur la théorie des ensembles et la logique des prédicats. Le langage logico-ensembliste de la méthode B événementiel supporte des notations ensemblistes usuelles, des symboles et des concepts mathématiques.

A.1 Le langage logico-ensembliste

nous présentons les notations relationnelles et ensemblistes que nous adoptons dans les formalisations dans les travaux de notre thèse.

A.1.1 Notations sur les relations

Nous choisissons les notations de la théorie des ensembles qui est un moyen privilégié par la plupart des scientifiques pour ses avantages multiples. Ces notations sont fondées sur la logique du premier ordre et ils permettent de manipuler des objets d'ordre élevé tels que les ensembles et les relations à n'importe quelle profondeur [J.-96b]. Nous fournissons dans le Tableau A.1 un résumé des notations de la théorie des ensembles. Les lettres R, f et E représentent respectivement une relation, une fonction et un ensemble. Nous représentons dans les Figures A.2, A.3, A.4, A.5, A.6, A.7, A.8 et A.9 quelques fonctions usuelles.

Symboles	Définitions	Symboles	Définitions
$\lhd\!\!\!\!-$	Soustraction du domaine	$\rightarrowtail\!\!\!\!\!\!\mapsto$	Injection partielle
$\lhd$	Restriction du domaine	$\rightarrowtail$	Injection totale
$\rhd$	Restriction du co-domaine	$\twoheadrightarrow\!\!\!\!\!\!\rightarrow$	Surjection partielle
$\rightarrow$	Fonction totale	$\twoheadrightarrow$	Surjection totale
$\longleftrightarrow$	Relation	$\rhd\!\!\!\!-$	Soustraction du co-domaine
$\rightarrowtail\!\!\!\!\!\!\twoheadrightarrow$	Bijective totale	$\lhd\!\!\!\!+$	Surcharge relationnel
$\rightarrow\!\!\!\!\!\!\rightarrow$	Fonction partielle	$ran(R)$	co-domaine de R
R^+	Fermeture transitive de R $\bigcup_{i\in\{1,2...\}} R^i$	$card(E)$	Cardinalité de l'ensemble E
$f(e)$	Image de e	$R[\{e\}]$	Image relationnelle ; donne l'ensemble des images
$\rightarrowtail\!\!\!\!\!\!\rightarrow$	Bijective partielle	$\mathbb{P}(EVT)$	Partie de l'ensemble E
R^{-1}	Relation inverse	$R;R$	Composition séquentielle
$id(E)$	Identité	R^+	Fermeture transitive
		R^*	Fermeture transitive réflexive

TABLE A.1 – Quelques opérateurs de la théorie des ensembles

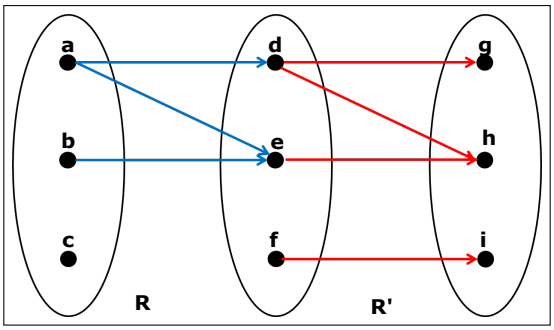

FIGURE A.1 – Illustration d'un exemple sur les relations

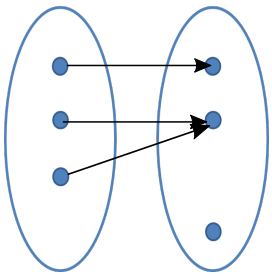

FIGURE A.2 – Fonction totale

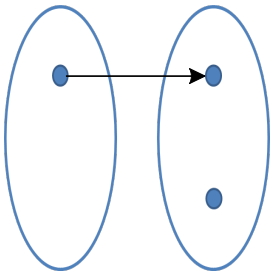

FIGURE A.3 – Injection totale

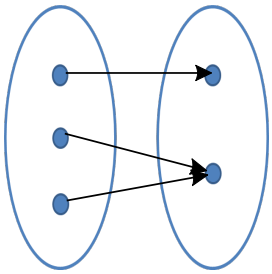

FIGURE A.4 – Surjection totale

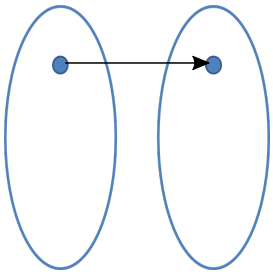

FIGURE A.5 – Bijection totale

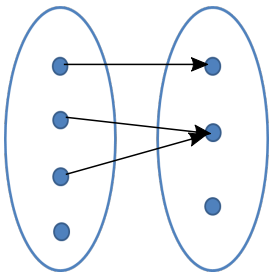

FIGURE A.6 – Fonction partielle

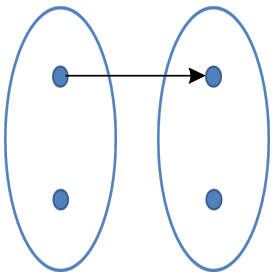

FIGURE A.7 – Injection partielle

Illustrations : considérons la Figure A.1 :
- $R = \{(a, d), (a, e), (b, e)\} = \{(a \mapsto d), (a \mapsto e), (b \mapsto e)\}$
- $dom(R) = \{a, b\}$
- $ran(R) = \{d, e\}$
- $R[\{b\}] = e$
- $R^{-1} = \{d \mapsto a, e \mapsto a, e \mapsto b\}$
- $id(source) = \{a \mapsto a, b \mapsto b, c \mapsto c\}$
- $R; R' = \{a \mapsto g, a \mapsto h, b \mapsto h\}$

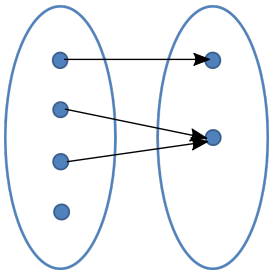

FIGURE A.8 – Surjection partielle

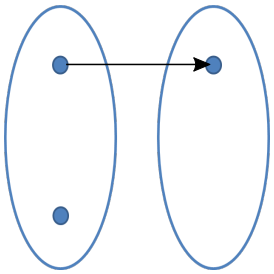

FIGURE A.9 – Bijection partielle

• $\{a, c\} \lhd R = \{a \mapsto d, a \mapsto e\}$
• $\{a, c\} \lhd\!\!\!- R = \{b \mapsto e\}$
• $R \rhd \{d, f\} = \{a \mapsto d\}$
• $R \rhd\!\!\!- \{d\} = \{a \mapsto e, b \mapsto e\}$
• $R1 \lhd\!\!\!- \{a \mapsto d, c \mapsto e, \} = \{a \mapsto d, b \mapsto e, c \mapsto e\}$
• Fermeture transitive : soient $E = \{1, 2, 3, 4\}$ et $R = \{1 \mapsto 1, 1 \mapsto 2, 2 \mapsto 1, 2 \mapsto 3\}$ alors $R^* = \{1 \mapsto 1, 1 \mapsto 2, 1 \mapsto 3, 2 \mapsto 1, 2 \mapsto 2, 2 \mapsto 3\}$
• Fermeture réflexive transitive : soient $E = \{1, 2, 3, 4\}$ et $R = \{1 \mapsto 1, 1 \mapsto 2, 1 \mapsto 3, 2 \mapsto 1, 2 \mapsto 2, 2 \mapsto 3, 3 \mapsto 3, 4 \mapsto 4\}$ alors $R^* = \{1 \mapsto 1, 1 \mapsto 2, 1 \mapsto 3, 2 \mapsto 1, 2 \mapsto 2, 2 \mapsto 3, 3 \mapsto 3, 4 \mapsto 4\}$

A.1.2 Propriétés les relations

Une relation binaire R est :
— réflexive si $\forall a \in E, (a, a) \in R$,
— non-réflexive si $\forall (a, b) \in E, (a, b) \in R \Longrightarrow a \neq b$,
— symétrique si $\forall (a, b) \in E, (a, b) \in R \Longrightarrow (b, a) \in R$,
— antisymétrique $\forall (a, b) \in E, (a, b) \in R \wedge (b, a) \in R \Longrightarrow a = b$
— asymétrique $\forall (a, b) \in E, (a, b) \in R \Longrightarrow \neg(b, a) \in R$,
— transitive $\forall a, b, c \in, (a, b) \in R \wedge (b, c) \in R \Longrightarrow (a, c) \in R$.

A.1.3 Les arbres

Un arbre est une structure de donnée. Il est constitué de nœuds qui sont organisés sous forme d'une hiérarchie. Dans ce qui suit nous énonçons le vocabulaire relié aux arbres.

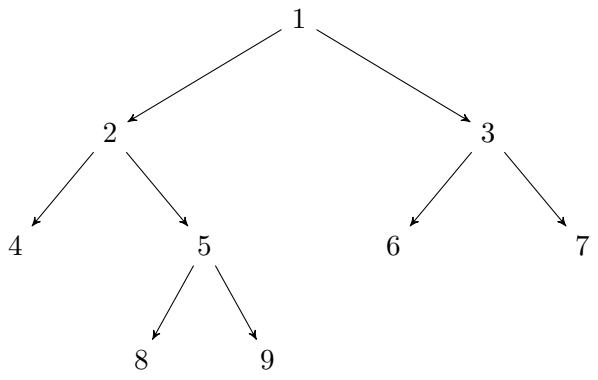

FIGURE A.10 – Exemple d'arbre

— La *racine* : c'est le nœud qui est situé à la tête de l'arbre. Un arbre admet une seule racine et un seul chemin à partir de la racine à n'importe quel nœud,

— *parent* : n'importe quel nœud, excepté la racine, qui possède au moins un arc descendant est appelé parent,
— *fils* : un nœud situé au dessous d'un nœud donné et qui est connecté par un arc dirigé vers le haut est appelé nœud fils,
— chaque nœud est soit une *feuille* ou un *nœud interne* : un nœud interne possède un ou plusieurs fils et une feuille ne possède pas de fils,
— les nœuds ayant le même parent sont appelés *descendance*,
— un *chemin* est une séquence de nœuds $n_0, n_1, ..., n_n$, tels qu'il y a un arc à partir d'un nœud à un nœud unique. Le chemin peut être seulement vers le bas, et il relie un nœud avec un descendant,
— la longueur d'un chemin est le nombre d'arcs dans le chemin,
— les descendants d'un nœud n sont tous les nœuds atteints par un chemin du nœud n jusqu'au nœud feuilles
— les ancêtres d'un nœud n sont tous les nœuds trouvés sur le chemin à partir de la racine au nœud n,
— la profondeur d'un nœud est le nombre d'arc à partir du nœud concerné jusqu'à la racine.

Quelques propriétés sur les arbres :

— tous les nœuds d'un arbre ont un et un seul parent sauf la racine,
— les nœuds frères ou sœurs sont des nœuds de même parent,
— racine= le seul nœud sans père,
— degré d'un nœud= nombre de ses enfant,
— profondeur d'un nœud= longueur du chemin entre la racine et ce nœud,
— hauteur d'un arbre= profondeur maximale de ses nœuds/hauteur de sa racine,
— niveau L dans un arbre= ensemble des nœuds de profondeur L,
— taille d'un arbre= nombre de ses nœuds.

Illustration : considérons l'arbre représenté par la Figure A.10, l'hauteur de la racine=3, la profondeur de la racine=0. La profondeur de la feuille 9 = 3, l'hauteur de la feuille 9=0. La profondeur du nœud 2=1, l'hauteur de la feuille 2=2.

A.2 La méthode B

B est un langage de spécification formelle inventé par Jean Raymond [J.-96b]. Il est basé sur la logique du 1^{er} ordre et la théorie des ensembles.

La méthode B est une méthode de développement formel des logiciels qui est inventée par Jean Raymond [J.-96b]. Elle est entièrement outillée, et elle a été appliquée avec succès dans plusieurs projets industriels [J. 99d], [S. 94], [F. 06b]. La méthode B événementiel [J.-07a] est une évolution de la méthode B classique [J.-96b]. Elle utilise la notion des événements pour la description des transitions entre les états.

A.2.1 Présentation de la méthode B événementiel (*Event-B*)

La méthode B événementiel est dotée de plusieurs aspects clés qui font d'elle une méthode privilégiée de développement, en effet elle est basée sur :
• l'utilisation de la théorie des ensembles comme notation de modélisation,
• l'utilisation du raffinement pour modéliser les systèmes à différents niveaux d'abstraction et
• l'utilisation des preuves mathématiques pour la vérification du modèle et des niveaux des raffinements.

A.2.2 Le modèle B événementiel

Un modèle B événementiel est composé par des *contextes* et des *machines*. Les contextes sont utilisés pour décrire et spécifier les éléments statiques du système étudié. Dans la machine,

quelques éléments statiques sont également spécifiés ainsi que le comportement dynamique du système. Un contexte peut être étendu par plusieurs autres contextes ; il peut également étendre plusieurs autres contextes. Une machine possède une visibilité sur les contextes. Elle peut être raffinée par plusieurs machines, cependant une machine en raffine au plus une seule.

A.2.2.1 Les contextes

Un contexte contient des *ensembles*, des *constantes* et des *axiomes*. Les propriétés des ensembles et des constantes sont décrites par des axiomes. Les axiomes sont des hypothèses qui sont utilisées dans les preuves. Les axiomes peuvent être marqués comme des théorèmes pour des propriétés dérivées.

A.2.2.2 La machine B événementiel

Une machine B possède une entête comprenant son nom. Elle est structurée sous forme de clauses (Figure A.11) :
• la clause SEES dans laquelle nous spécifions les contextes vus par ma machine.
• la clause VARIABLES contient la liste des variables,
• la clause INVARIANTS exprime le typage des VARIABLES et des propriétés d'invariance,
• la clause THEOREM contient des propriétés qui peuvent être déduites des propriétés d'invariance mais qui ne constituent pas forcément des invariants inductifs, cette clause peut également contenir des propriétés que nous souhaitons prouver pour les utiliser dans la preuve des invariants du modèle,
• la clause INITIALISATION dans laquelle les valeurs initiales des variables sont attribuées,
• la clause EVENTS contient la liste des événements.

```
MACHINE
/*nom de la machine*/
SEES
/*listes des contextes "vus" par la machine*/
VARIABLES
/*liste des variables
INVARIANT
/*propriètès d'invariance du système et
typage des variables*/
THOREMS
/*liste des théorèmes de la machine
EVENTS
/*liste des événements de la machine*/
```

FIGURE A.11 – Machine abstraite B

Les événements B. La notion d'événement B est similaire aux actions de [Ral94] ou aux commandes gardées de Dijkstra [Eds75]. Un événement est constitué de trois éléments : *i)* son nom, *ii)* les gardes, que nous notons $G(t, v)$, sont des prédicats qui définissent les conditions nécessaires pour son déclenchement et *iii)* les substitutions généralisées, que nous notons $S(t, v)$, où v représente les variables et t représente les paramètres de l'événement. Les substitutions généralisées décrivent comment les variables sont modifiées après l'occurrence de l'événement.
Un événement peut avoir une des trois formes qui sont illustrées dans le Tableau A.2. La première forme est la plus générale, la deuxième forme est utilisée lorsque l'événement ne possède pas de paramètres et la troisième forme est utilisée si, en plus, la garde est toujours vraie.

Forme 1	Forme 2	Forme 3
ANY	WHEN	BEGIN
t	$G(v)$	$S(v)$
WHEN	THEN	END
$G(t, v)$	$S(v)$	
THEN	END	
$S(t, v)$		
END		

TABLE A.2 – Les formes possibles d'un événement en B événementiel

Les substitutions généralisées : dans les spécifications B, les instructions sont exprimées dans le langage des substitutions généralisées. C'est un langage qui est basé sur la théorie des transformateurs de prédicats, il a été introduit par Dijkstra [E. 76] et il est inspirée par la logique de Hoare [C. 78]. Les substitutions généralisées en B peuvent être réécrites en termes de prédicats avant/après. Les prédicats portent sur les variables d'état de la machine considérée.

Modèle d'exécution des événements dans le modèle B événementiel : un événement se déclenche si sa garde est évaluée à vraie. Plusieurs gardes d'événements peuvent être évaluées à *vrai* simultanément, cependant, un seul événement peut se déclencher. Le choix de l'événement qui se déclenche est non déterministe. Un événement définit un changement d'état possible de la machine.

A.2.3 Sémantique de B événementiel : les obligations de preuves

La sémantique formelle de B événementiel est définie par un ensemble de propriétés logiques du modèle. La syntaxe de B événementiel et la structure du modèle permettent aux outils de générer automatiquement des *obligations de preuve* (OPs). Les obligations de preuve sont une aide au processus de vérification de la correction d'un modèle B. En effet chaque obligation de preuve est une formule mathématique qui doit être démontrée. La génération des obligations de preuve s'appuie sur le calcul de la plus faible pré-condition. Il s'agit de vérifier que l'invariant est respecté aussi bien par l'initialisation que par les événements de la machine qui modifie son état.
Dans ce qui suit, nous présentons les principales obligations de preuve.

• *Bonne définition* (*well-definedness*) *WD* : les obligations de preuve WD assurent que les prédicats, les expressions et les affectations sont bien définies.
• *Théorèmes prouvables* (*provable theorem*) *THM* : les obligations de preuve THM assurent que les théorèmes qui sont déclarés dans le contexte ou dans la machine sont prouvables. La validité d'un théorème doit être prouvée à partir des axiomes, des invariants déclarés avant ce théorème. Ces théorèmes sont importants pour simplifier quelques preuves.
• *Préservation de l'invariant* (*invariant preservation*) *INV* : l'invariant du modèle est une propriété qui doit être vérifiée initialement et que le système doit préserver quelle que soit son évolution (i.e après le déclenchement de chaque événement du système).
• *Faisabilité* (*feasibility*) *FIS* : tous les événements d'une machine B événementiel doivent être toujours faisables. Les obligations de preuve FIS et INV expriment la cohérence logique (*logical consistency*) du modèle. Les obligations de preuve de l'INV garantissent que le *post*-état demeure dans les états légaux. Les obligations FIS garantissent que tous les états satisfaisant la garde sont reliés par le prédicat avant-après a au moins un post-état.
Considérons un modèle M avec des variables v, qui a une visibilité sur un contexte C possédant les ensembles s et les constantes c. Les propriétés des constantes sont notées $P(s, c)$ et l'invariant est noté $I(s, c, v)$. Soit E un événement de M avec les gardes $G(s, c, v)$ et le prédicat $R(s, c, v, v')$.

Nous devons d'abord exprimer que, sous les propriétés $P(s, c)$, l'invariant $I(s, c, v)$ et la garde

$G(s, c, v)$, le prédicat avant-après donne au moins une valeur après v' qui est définie par le prédicat avant-après $R(s, c, v, v')$. C'est l'obligation de preuve de faisabilité FIS. Nous exprimons ensuite que l'invariant est préservé. C'est l'obligation de preuve d'invariance INV. Formellement ces deux obligations de preuve s'expriment comme suit :

$P(s,c) \ \wedge \ I(s,c,v) \ \wedge \ G(s,c,v) \Longrightarrow \exists v'.R(s,c,v,v')$	FIS
$P(s,c) \ \wedge \ I(s,c,v) \ \wedge \ G(s,c,v) \ \wedge \ R(s,c,v,v') \Longrightarrow I(s,c,v')$	INV

Pour l'événement d'initialisation il y a une loi spéciale. Soit $RI(s, c, v')$ le prédicat après de la substitution généralisée associée à cet événement. Les obligations à prouver notées *INI_FIS* et *INI_INV*, sont formellement exprimées comme suit :

$P(s,c) \Longrightarrow \exists v'.RI(s,c,v)$	INI_FIS
$P(s,c) \ \wedge \ RI(s,c,v) \Longrightarrow I(s,c,v')$	INI_INV

A.2.4 Raffinement d'une machine B événementiel

Le raffinement est une notion clé dans la méthode B événementiel. Il permet de développer le système de manière incrémentale en partant du modèle abstrait qui constitue une spécification du système. Les détails du système sont rajoutés de façon progressive dans chaque raffinement.

Le raffinement d'une machine abstraite est une transformation produisant une machine qui conserve la même interface (même événements) et le même comportement que la machine abstraite initiale. La structure d'une machine raffinée (Figure A.12) est similaire à celle d'une machine abstraite avec l'ajout de la clause REFINES qui indique le nom de la machine raffinée. La clause qui contient le nom de la machine est nommée REFINEMENT au lieu de MACHINE. La clause VARIABLES contient les variables qui étaient dans la machine abstraite et que nous pouvons conserver dans le raffinement, ainsi que les variables introduites dans le nouveau raffinement. Dans la clause INVARIANT, le typage des nouvelles variables est exprimé ainsi que le lien entre les variables abstraites et les variables introduites dans le raffinement. Dans la clause VARIANT une expression arithmétique est exprimée et elle permet d'empêcher la divergence des nouveaux événements introduits.

Le raffinement permet d'une part, de reformuler la machine en une expression de plus en plus concrète, et d'autre part, de l'enrichir en y ajoutant de nouvelles données et invariants. L'état d'une machine abstraite M est lié à l'état d'une machine concrète par le biais d'un *invariant de collage* $J(s, c, v, w)$ qui est associé à la machine concrète N, où v sont les variables de la machine abstraite et w sont les variables de la machine concrète.

A.2.4.1 Raffinement des événements existants

Chaque événement de la machine abstraite est raffiné par un ou plusieurs événements. Soit $E(v)$ un événement abstrait ayant une garde $G(t, s, c, v)$ et un prédicat avant-après $R(s, c, v, v')$, et $F(w)$ un événement concret ayant une garde $H(s, c, w)$ et un prédicat avant-après $S(s, c, w, w')$.

Intuitivement, l'événement concret $F(w)$ raffine l'événement abstrait $E(v)$ chaque fois que l'invariant de collage $J(v, w)$ est vrai : *i)* la garde de l'événement concret $F(w)$ est plus forte que la garde de l'événement abstrait $E(v)$ et *ii)* pour chaque exécution possible de $F(w)$ il y a une exécution correspondante de $E(v)$ qui simule $F(w)$ tels que l'invariant de collage reste vrai après l'exécution des deux événements.
Formellement il faut prouver 3 obligations de preuves du raffinement que nous illustrons dans le Tableau A.3.

REFINEMENT
/*nom de la machine*/
REFINES
/*nom de la machine*/
SEES
/*nom de la machine raffinée*/
VARIABLES
/*liste des variables d'état du modèle
INVARIANT
/*propriètès d'invariance du système*/
VARIANT
/*variant du système*/
THEOREMS
/*liste des théorèmes de la machine
EVENTS
/*liste des événements de la machine*/

FIGURE A.12 – Machine raffinée

$P(s,c) \ \wedge \ I(s,c,v) \ \wedge \ J(s,c,v,w) \ \wedge \ H(s,c,w)$ $\Longrightarrow \exists w'.S(s,c,w,w')$	FIS_REF
$P(s,c) \ \wedge \ I(s,c,v) \ \wedge \ J(s,c,v,w) \ \wedge \ H(s,c,w)$ $\Longrightarrow G(s,c,v)$	GRD_REF
$P(s,c) \wedge I(s,c,v) \wedge J(s,c,v,w) \wedge H(s,c,w) \wedge S(s,c,w,w')$ $\Longrightarrow \exists v'.(R(s,c,v,v') \ \wedge \ J(s,c,v',w'))$	INV_REF

TABLE A.3 – Obligations de preuve de raffinement

L'obligation de preuve FIS_REF, exprime que l'événement raffiné est faisable. Les obligations de preuve GRD_REF et INV_REF expriment un raffinement correcte de l'événement concret par rapport à son événement abstrait.

A.2.4.2 Introduction de nouveaux événements dans le raffinement

De nouveaux événements peuvent être introduits dans la machine raffinée. Le mécanisme de raffinement diffère légèrement du raffinement des événements existants dans la machine abstraite que nous avons décrit. Pour ces nouveaux événements deux contraintes doivent être vérifiées : *i)* chaque nouveau événement raffine un événement implicite *skip*, *ii)* les nouveaux événement ne doivent pas diverger (il ne doivent pas prendre le contrôle indéfiniment). Des obligations de preuves sont associées à ces contraintes.

Soit M un modèle abstrait qui a une visibilité sur le contexte C. Ce modèle est raffiné en un modèle concret N qui a une visibilité sur le contexte D (qui est le raffinement du contexte C). Supposons que dans le modèle raffiné N nous avons un nouvel événement avec une garde $H(s,c,w)$ et un prédicat avant-après $S(s,c,w,w')$. La première contrainte (raffinement du *skip*) mène aux obligations de preuve de raffinement indiquées dans le Tableau A.4.

Le deuxième contrainte (la non-divergence des nouveaux événements) impose de définir un variant $V(s,c,w)$. Il faut prouver que chaque nouvel événement décrémente ce variant. Nous représentons dans le Tableau A.5 l'obligation de preuve de la non divergence.

$P(s,c) \ \wedge \ I(s,c,v) \ \wedge \ J(s,c,v,w) \ \wedge \ H(s,c,w)$ $\implies \exists w'.S(s,c,w,w')$	FIS_REF
$P(s,c) \wedge I(s,c,v) \wedge J(s,c,v,w) \wedge H(s,c,w) \wedge S(s,c,w,w')$ $\implies J(s,c,v',w'))$	INV_REF

TABLE A.4 – Obligations de preuve de nouveaux événement raffinement

$P(s,c) \wedge I(s,c,v) \wedge J(s,c,v,w) \wedge H(s,c,w) \wedge S(s,c,w,w')$ $\implies V(s,c,w) \ \in \mathbb{N} \ \wedge \ V(s,c,w') \ < \ V(s,c,w)$	WFD_REF

TABLE A.5 – Obligation de preuve de la non-divergence des nouveaux événements

A.3 L'environnement B événementiel

A.3.1 La plateforme Rodin

La plateforme Rodin (Figure A.13) est un environnement de développement intégré qui est basé sur *Eclipse* pour le B événementiel. L'environnement fournit aux spécifieurs un support efficace pour l'édition des spécifications, le raffinement des modèles, la génération et la décharge des OPs.

A.3.1.1 La plateforme Eclipse

La plateforme Eclipse fournit les outils basiques pour la construction d'un environnement de développement intégré. Elle est facilement personnalisable pour supporter n'importe quel processus de développement particulier.

A.3.1.2 Les plug-ins noyaux

Les plug-ins noyaux fournissent les fonctionnalités basiques de modélisation et de preuves avec le B événementiel. Ils sont composés d'un ensemble de plug-ins pour la sauvegarde du modèle, la vérification, la gestion des preuves et l'interface utilisateur. Nous distinguons les plug-ins suivants :

• **Core** : ce plug-in fournit des API bas niveau pour manœuvrer les modèles B événementiel avec une gestion de base de données. Tous les éléments liés aux modèles B événementiel sont sauvegardés dans une base de données Rodin ou des fichiers XML. Ce plug-in fournit également un générateur de projets incrémental pour planifier le vérificateur statique, le générateur des obligations de preuve et le prouveur,

• **Le vérificateur statique** : ce plug-in fournit des API pour vérifier les modèles en B événementiel. Le vérificateur statique est appelé une fois que le modèle B événementiel est sauvegardé. Les formules mathématiques sont statiquement vérifiées pour qu'elles soient significatives,

• **Le générateur des obligations de preuves** : ce plug-in génère automatiquement les OPs pour les modèles B événementiel qui ne comportent pas des erreurs,

• **Prouveur** : ce plug-in fournit des APIs pour décharger les OPs. Il contient un gestionnaire de preuves et un ensemble de règles de déduction,

• **Interface utilisateur de modélisation** : ce plug-in fournit une interface graphique utilisateur pour écrire et éditer les modèles en B événementiel,

• **l'interface utilisateur de preuves** : ce plug-in fournit une interface graphique utilisateur pour l'affichage, la gestion et la décharge des preuves interactives.

A.3.1.3 Les plug-ins externes

Les plug-ins externes sont tous les autres plug-ins qui peuvent être utilisés dans la plateforme Rodin. Certains d'entre eux ont été développés au cours du projet Rodin, tandis que d'autres

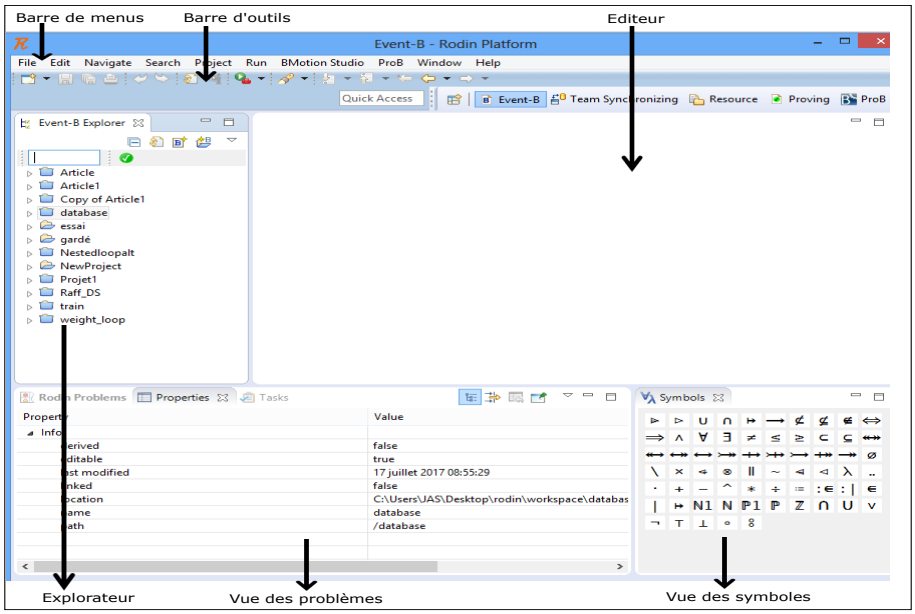

FIGURE A.13 – Interface Rodin

ont été développés plus tard. Parmi les plug-ins les plus connus citons l'éditeur Camille [Jen10] qui est un éditeur de texte simple pour le B événementiel. Il est similaire à celui des éditeurs de langue de programmation classique. Il complète l'éditeur structuré standard de Rodin. Les outils de vérification : le prouveur AtelierB [Cle09] qui est un prouveur puissant, Isabelle [Mat11] qui exporte les OPs dans la théorie d' Isabelle/HOL, le prouveur SMT. Les animateurs Brama [Ati09], AnimB [C. 12], ProB [LB03], [M. 08]. L'animateur UML-B pour les machines à états [Vit09] qui fournit une animation des machines à état d'UML-B State en utilisant le ProB.

Au sein de nos travaux, nous avons utilisé comme animateur l'explorateur de modèles ProB que nous présentons brièvement ci-dessous.

A.3.2 L'explorateur des modèles ProB

ProB est un animateur permettant une animation automatique des spécifications en B (Figure A.14). C'est un solveur de contraintes et un explorateur de modèles (*model-checker*) pour la méthode B qui peut être utilisé pour la vérification automatique des spécifications pour un large rang d'erreurs (Figure A.15). Il supporte plusieurs techniques de validation telles que la simulation et la vérification. En effet, il permet la simulation avec une étude exhaustive de toutes les exécutions possibles du modèle dans le cas où l'espace d'états est fini sinon il permet une simulation non exhaustive pour les machines B avec des espaces des états qui sont infinis. ProB est utilisé par plusieurs compagnies telles que *Siemens Alstom, Thales* pour la validation des données des propriétés complexes pour la sécurité des systèmes critiques. En effet, il peut être exploité pour la recherche des violations de l'invariant, la recherche des blocages, la recherche des violations des obligations de preuves pour les modèles et la génération de cas de tests (*test-case*). En outre, ProB peut être utilisé également pour la vérification automatique du raffinement des spécifications en B [LB03]. Nous fournissons plus de détails sur la validation des propriétés dans la section suivante.

Quelques propriétés importantes du système étudié

Les propriétés de sûreté : ces propriétés définissent les conditions dans lesquelles le système doit fonctionner. Elles expriment les conditions néfastes à éviter *(something bad will not happen*

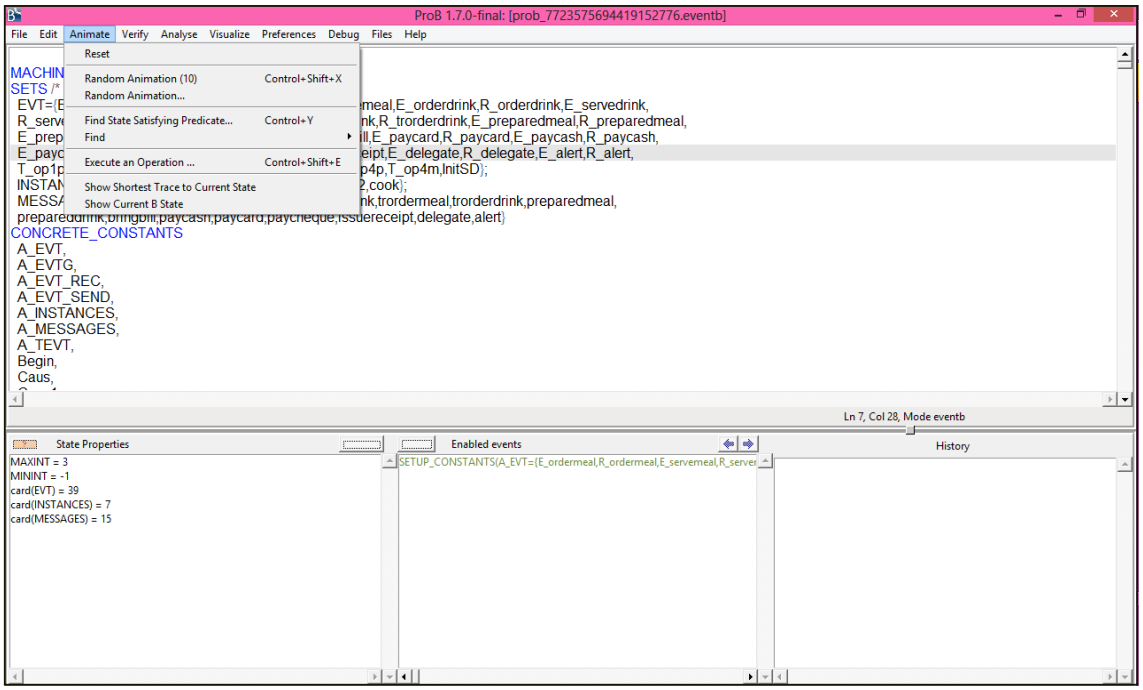

FIGURE A.14 – Interface ProB (animation)

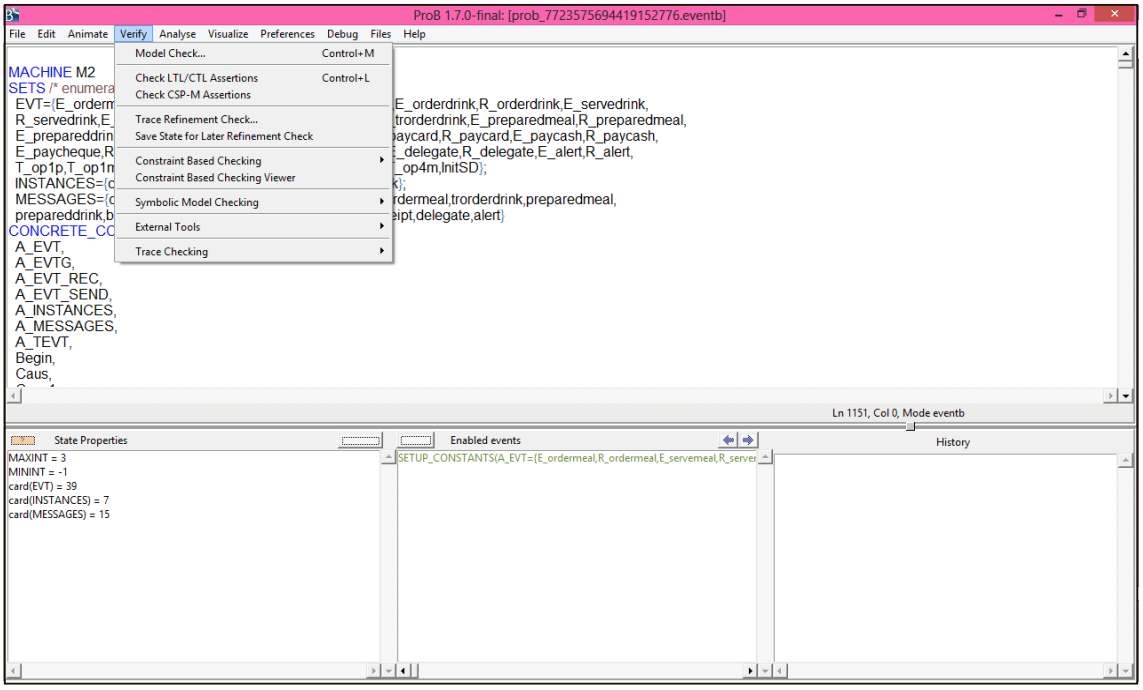

FIGURE A.15 – Interface ProB (vérification)

during an execution system). Parmi les propriétés de sûreté nous citons la préservation de l'invariant qui est une condition qui doit être maintenue en permanence sur les variables d'états. La préservation de l'invariant veut dire que les actions des événements ne violent pas les invariants.

Les propriétés de vivacité : Les propriétés de vivacité expriment les bonnes conditions à remplir *(eventulally something good must happen during an execution system).*
Pour montrer que les systèmes sont vivants. Il faut prouver que les spécifications en B associées ne se bloquent pas (l'absence de blocage (*deadlock freeness*)), elles ne divergent pas (non-divergence) et elles sont préservatrices de la déclenchabilité (*enabledness preserving*).

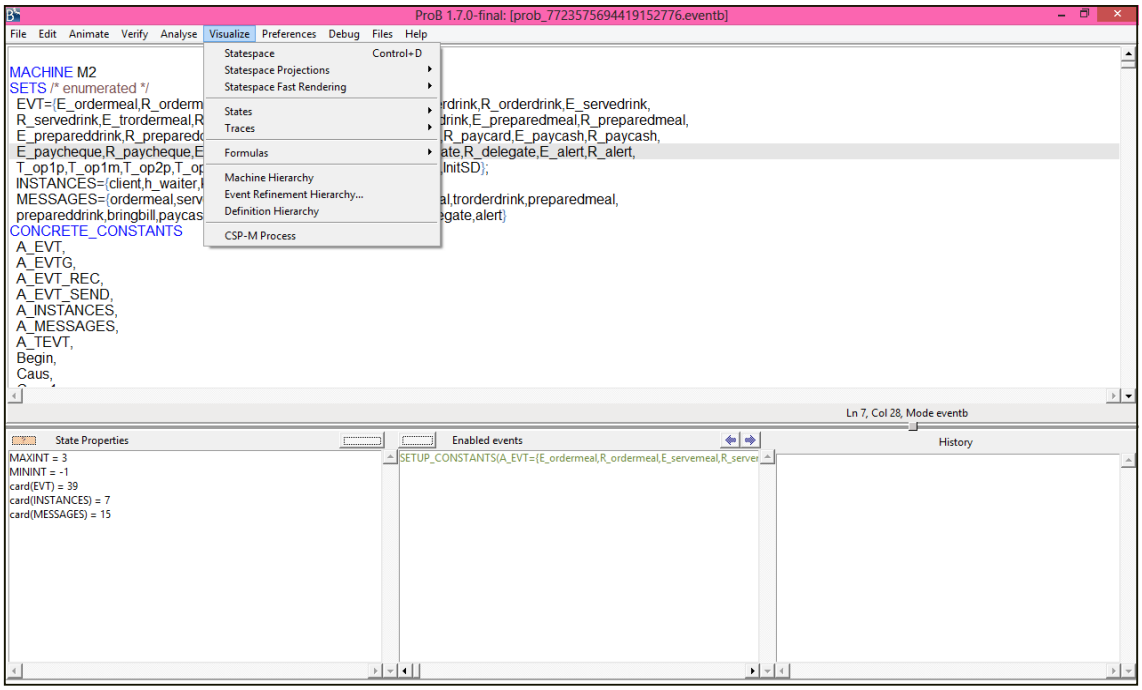

FIGURE A.16 – Interface ProB (visualisation)

Modalité : les modalités sont des propriétés sur les traces d'états. Un des formalismes les plus connus est la logique linéaire temporelle (LTL). En B événementiel, il existe plusieurs façons de spécifier quelques modalités [L.M98]. Par exemple nous pouvons exprimer des propriétés sur le futur immédiat et le futur imprécis des états du système : $\Box P$ qui signifie que sur chaque trace, tous les états satisfont le prédicat P, et $\diamondsuit\ P$ qui signifie que sur chaque trace, il existe des états qui satisfont P. On a aussi la notation $\bigcirc\ P$ pour un état, qui signifie que P est vrai dans l'état suivant.

A.3.2.0.1 Vérification des propriétés avec le prouveur des théorèmes Rodin. Les outils de B génèrent des obligations de preuve pour assurer ce qui suit.

- la vérification de la cohérence (*consistency checking*) : la cohérence d'une machine est établie tout en présumant que les actions associées avec chaque événement modifient les variables telles que les invariants sont préservés sous les hypothèses que les invariants sont vérifiés initialement et que les gardes sont vraies. Les obligations de preuve générées de la vérification de la cohérence prouvent que la machine est cohérente par rapport aux invariants.
- La vérification du raffinement (*refinement checking*) : les obligations de preuve déchargées de la vérification du raffinement assurent que les invariants de collage sont préservés par les actions des événements dans le raffinement.

Pour un système M donné, nous désignons par D les déclarations, x ses variables, $I(x)$ l'invariant et E_i ses événements. La condition de non-blocage s'exprime comme suit.

$$D\ \wedge\ I(x) \Longrightarrow \bigvee_i grd(E_i)$$

avec i parcourant l'ensemble des événements. En raffinant le système M en un système N. Soit y les variables du système N, $J(y)$ l'invariant de collage, E_i^a sont les événements du système abstraits, E_i^a sont les mêmes événements concrets et N_i^c sont les nouveaux événements

La préservation de cette vivacité par un raffinement est donnée, comme suit.
$D\ \wedge\ I(x)\ \wedge\ J(y)\ \wedge\ \bigvee_i\ grd(E_i^a) \Longrightarrow\ \bigvee_i\ grd(E_i^c)\ \vee\ \bigvee_j\ grd(N_j^c)$

Prouver la non-divergence nécessite de prouver que les nouveaux événements ne prennent pas le contrôle indéfiniment. Comme nous l'avons mentionné dans la Section A.2.4 ceci est garanti par la définition, dans la clause VARIANT de la machine concrète, d'une expression arithmétique strictement positive qui est décrémentée à chaque fois qu' un nouvel événement se déclenche.
La préservation de la déclenchabilité revient à vérifier qu'à chaque fois qu'un événement est déclenchable à un niveau abstrait alors un événement correspondant est déclenchable au niveau concret.

A.3.2.0.2 Vérification des propriétés avec ProB. ProB est basé sur Prolog. Il permet la vérification automatique de la cohérence (*consistency*) des machines B. Avec ProB, nous pouvons découvrir des problèmes de spécification. Il nous alerte aussitôt que le problème est découvert et il présente la trace la plus courte (celle qui figure dans les états explorés) qui mène de l'état initial jusqu'à l'erreur. L'explorateur de modèles détecte aussi quand tous les états ont été explorés. Par conséquent il peut être utilisé pour garantir formellement l'absence d'erreurs dans la spécification. ProB offre plusieurs aspects de visualisation conviviale facilitant ainsi l'analyse et la compréhension de la spécification B (Figure A.16). Lors de la vérification d'une spécification, le processus d'animation peut être paramétré à travers plusieurs préférences ou commandes selon le besoin de l'utilisateur.

A.3.2.0.3 Vérification des propriétés temporelles. ProB peut être utilisé pour la vérification des propriétés temporelles linéaires (*linear temporal logic (LTL)*) et des propriétés de la logique temporelle arborescente (*computation tree logic (CTL)).* En effet, ProB supporte le formalisme LTL[e], qui est une version étendue du LTL. Contrairement au LTL classique qui supporte uniquement les états, LTL[e] fournit un support pour les propositions sur les transitions (ou les événements). En pratique, pour écrire des propositions sur les transitions nous utilisons les constructeurs $e(...)$ et [...]. L'explorateur des modèles LTL pour le ProB supporte aussi le formalisme *Past-LTL[e].*

A.3.2.0.4 Vérification du raffinement. Le raffinement considéré par ProB est différent du raffinement B. En effet ProB ne prend pas en compte l'invariant de collage. Techniquement parlant, un algorithme est proposé qui parcourt les états des deux systèmes, en construisant une structure de collage R entre eux. Dans le cas de succès de l'algorithme, R lie chaque état initial concret à un ensemble d'états abstraits et la condition de simulation est vérifiée pour chaque couple d'états liés. Ils vérifient aussi le raffinement des traces entre les deux systèmes de transitions étiquetées.

Avec ProB la vérification des propriétés est très rapide ce qui permet un gain de temps. Pour certaines propriétés, nous optons pour leurs vérifications avec ProB plutôt qu'avec Rodin, par exemple la propriété de non-blocage est une propriété qui est assez lourde à exprimer avec Rodin (la disjonction de toutes les gardes de tous les événements) et elle est délicate à prouver.

Bibliographie

[Da07] Dan Haitao and M. Hierons Robert and Steve Counsell. Thread-Based Analysis of Sequence Diagrams. In *Formal Techniques for Networked and Distributed Systems - FORTE 2007, 27th IFIP WG 6.1 International Conference Tallinn Estonia Proceedings*, pages 19–34, 2007.

[Gr07] Gregor Engels and Christian Soltenborn and Heike Wehrheim. Analysis of UML Activities Using Dynamic Meta Modeling. In *Proceedings of the 9th IFIP WG 6.1 International Conference on Formal Methods for Open Object-based Distributed Systems*, pages 76–90. Springer-Verlag, 2007.

[H.02] H. Ledang. *Traduction Systématique de Spécifications UML Vers B.* Thése d'université, LORIA - Nancy 2, 2002.

[Ja01] Janette Cardoso and Christophe Sibertin-Blanc. Ordering Actions in Sequence Diagrams of UML. In *23rd International Conference on Information Technology Interfaces*, pages 3–14. University Computing Centre University of Zagreb, 2001.

[Ol07] Olfa Mosbahi and Jacques Jaray and Samir Ben Ahmed. Spécification et Vérification des Propriétés de Vivacité en B Événementiel. In *6ème Colloque Francophone sur la Modélisation des Systèmes Réactifs - MSR 2007*, 2007.

[Yo06] Youcef Hammal. Branching Time Semantics for UML 2.0 Sequence Diagrams. *Lecture Notes in Computer Science : Formal Techniques for Networked and Distributed Systems -FORTE*, pages 259–274, 2006.

[A. 96] A. Nancy Lynch and W. Frits Vaandrager. Forward and Backward Simulations, II : Timing-Based Systems. *Inf. Comput.*, 128(1) :1–25, 1996.

[A. 02a] A. David and M. O Möller. From HUppaal to Uppaal : A Translation from Hierarchical Timed Automata to Flat Timed Automata. In *Fundamental Approaches to Software Engineering : 5th International Conference, FASE*, 2002.

[A. 02b] A. Knapp and S. Merz. Model Checking and Code Generation for UML State Machines and Collaborations. In G. Schellhorn and W. Reif, editors, *FM-TOOLS 2002 : 5th Workshop on Tools for System Design and Verification.* Institut für Informatik Universität Augsburg, 2002.

[A. 14] A. Eerke Boiten. Formal Aspects of Computing. volume 26, pages 305–317, 2014.

[Ale02] Alexander Knapp and Stephan Merz and Christopher Rauh. Model Checking Timed UML State Machines and Collaborations. *Formal Techniques in Real-Time and Fault-Tolerant Systems*, 2002.

[Ale04] Alessandra Cavarra and Juliana Küster Filipe. Formalizing Liveness-Enriched Sequence Diagrams Using ASMs. In *Abstract State Machines 2004. Advances in Theory and Practice, 11th International Workshop, Lutherstadt Wittenberg, Germany, Proceedings*, pages 62–77, 2004.

[Ale06] Alexander Knapp and Jochen Wuttke. Model Checking of UML 2.0 Interactions. In Thomas Kühne, editor, *Models in Software Engineering*, pages 42–51. Springer, 2006.

[Ale14] Alexander Knapp and Harald Störrle. Efficient Representation of Timed UML 2 Interactions. In *System Analysis and Modeling : Models and Reusability - 8th International Conference, SAM 2014, Valencia, Spain. Proceedings*, pages 110–125, 2014.

[Ama94] Amar Bouali and Stefania Gnesi and Salvatore Larosa. The Integration Project for the JACK Environment, 1994.

[Ati09] Atif Mashkoor and Jean-Pierre Jacquot and Jeanine Souquières. Transformation Heuristics for Formal Requirements Validation by Animation. *In 2nd International Workshop on the Certification of Safety-Critical Software Controlled Systems, York, UK*, 2009.

[B. 96] B. Rumpe. Formale Methodik des Entwurfs Verteilter Objekt Orientierter Systeme. PhD thesis, Institut für Informatik, Technische Universität München, 1996.

[B. 02] B. Aredo Demissie. A Framework for Semantics of UML Sequence Diagrams in PVS. *Journal of Universal Computer Science (JUCS)*, pages 674–697, 2002.

[BB02] H. Ruiz Barradas and D. Bert. Specification and Proof of Liveness Properties under Fairness Assumptions in B Event Systems. In *Integrated Formal Methods (IFM 2002)*, pages 360–379. LNCS 2335, Springer-Verlag, 2002.

[BB06] Héctor Ruiz Barradas and Didier Bert. Propriétés Dynamiques avec Hypothèses d'équité en B Événementiel. *Technique et Science Informatiques, RSTI*, 25 :73–102, 2006.

[Boi14] Eerke A. Boiten. Introducing Extra Operations in Refinement. *Formal Aspects of Computing*, 26(2) :305–317, 2014.

[But09] Michael Butler. Decomposition Structures for Event-B. In Leuschel Michael and Wehrheim Heike, editors, *Integrated Formal Methods*, pages 20–38. Springer Berlin Heidelberg, 2009.

[C. 78] C. A. R. Hoare. Communicating Sequential Processes. *Commun. ACM*, 21(8) :666–677, 1978.

[C. 05a] C. Eichner and H. Fleischhack and R. Meyer and U. Schrimpf and C. Stehno. Compositional Semantics for UML 2.0 Sequence Diagrams Using Petri Nets. *Lecture Notes in Computer Science*, 3530 :133–148, 2005.

[C. 05b] C. Sibertin-Blanc and O. Tahir and J. Cardoso. Interpretation of UML Sequence Diagrams as Causality Flows. In *Advanced Distributed Systems 5th Int. School and Symposium (ISSAD)*, number 3563, pages 126–140. Acta Press, 2005.

[C. 12] C. Métayer. B Model Animator. 2012.

[Chi05] Chien An Chen and Sara Kalvala and Jane Sinclair. Race Conditions in Message Sequence Charts. In *3rd Asian Symposium on Programming Languages and Systems (APLAS'05)*, pages 195–211, 2005.

[Chr06] Christophe Sibertin-Blanc and Omar Tahir. From UML1.x to UML 2.0 Semantics for Sequence Diagrams. In F. F. Ramos and R. V. Lrios and H. Unger, editor, *IEEE International Symposium and School on Advance Distributed Systems (ISSADS), Mexico (Mexique)*. IEEE, 2006.

[Cle09] Clearsy. Atelier B Interactive Prover Reference Manual, version 4.0. Technical report, 2009.

[D. 09] D. Fischbein and V. A. Braberman and S. Uchitel. A Sound Observational Semantics for Modal Transition Systems. *Lecture Notes in Computer Science*, 5684 :215–230, 2009.

[Dav08] David Harel and Shahar Maoz. Assert and Negate Revisited : Modal Semantics for UML Sequence Diagrams. In *Software and System Modeling*, volume 7(2), pages 237–253, 2008.

[E. 76] E. W. Dijkstra. A Discipline of Programming. In *Prentice Hall PTR Upper Saddle River NJ USA*, 1976.

[E. 86] E. Brinksma and G. Scollo. Formal Notions of Implementation and Conformance in LOTOS. *Tech. Report INF, Twente University of Technology, Department of Informatics Enschede Netherlands*, pages 86–13, 1986.

[Eds75] Edsger W. Dijkstra. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *Commun. ACM*, 18(8) :453–457, aug 1975.

[Eri99] Eric Meyer and Jeanine Souquières. A Systematic Approach to Transform OMT Diagrams to a B Specification. In *Formal Methods World Congress on Formal Methods in the Development of Computing Systems Toulouse France Proceedings Volume I*, pages 875–895, 1999.

[F. 03] F. Colin Snook and Kim Sandström. Using UML-B and U2B for Formal Refinement of Digital Components. In *Forum on Specification and Design Languages, FDL 2003, Frankfurt, Germany, Proceedings*, pages 505–515, 2003.

[F. 06a] F. Colin Snook and J. Michael Butler. UML-B : Formal Modeling and Design Aided by UML. *ACM Trans. Softw. Eng. Methodol.*, 15 :92–122, 2006.

[F. 06b] F. Patin and G. Pouzancre and D. Sabatier and S. Hauvespre and P. Sauvage. Utilisation de la Méthode Formelle B pour un Système SIL3 : la Commande des Portes Palière sur La ligne 13 du Métro Parisien. 2006.

[Fam13] Fama Diagne. *Proving Dynamic Properties in B.* PhD thesis, Institut National des Télécommunications, 2013.

[Fam14] Fama Diagne and Amel Mammar and Marc Frappier. A Tool for Verifying Dynamic Properties in B. In Dimitra Giannakopoulou and Gwen Salaün, editors, *Software Engineering and Formal Methods*, pages 290–295. Springer International Publishing, 2014.

[Fat15] Fatma Dhaou and Inès Mouakher and Christian Attiogbé and Khaled Bsaies. Extending Causal Semantics of UML2.0 Sequence Diagram for Distributed Systems. *ICSOFT-EA 2015 - Proceedings of the 10th International Conference on Software Engineering and Applications, Colmar, Alsace, France*, pages 339–347, 2015.

[Fat16] Fatma Dhaou and Inès Mouakher and Christian Attiogbé and Khaled Bsaïes. Refinement of UML2.0 Sequence Diagrams for Distributed Systems. In *Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016) - Volume 1 : ICSOFT-EA Lisbon Portugal*, pages 310–318, 2016.

[Fat17] Fatma Dhaou and Inès Mouakher and Christian Attiogbé and Khaled Bsaïes. A Causal Semantics for UML2.0 Sequence Diagrams with Nested Combined Fragments. In *ENASE 2017 - Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering, Porto, Portugal*, pages 47–56, 2017.

[Fat18] Fatma Dhaou and Inès Mouakher and Christian Attiogbé and Khaled Bsaïes. Guard Evaluation and Synchronization issues in Causal Semantics for UML2.X Sequence Diagrams. In *ENASE 2018 - Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering, Madere, Portugal*, 2018.

[Fré05] Frédéric Gervais and Marc Frappier2 and Régine Laleau3. Vous Avez dit Raffinement ? *Rapport Technique GRIL Université de Sherbrooke*, 2005.

[Gab08] Gabor Huszerl and Hélène Waeselynck (ed.) and Zoltàn Égel and Andràs Kövi and Zoltàn Micskei. Refined Design and Testing Framework Methodology and Application Results. 2008.

[Ger01] Gerson Sunyé and Damien Pollet and Yves Le Traon and Jean-Marc Jézéquel. Refactoring UML models. In *The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, pages 134–148, 2001.

[Gra90] Grady Booch. *Object-Oriented Design with Applications.* Benjamin/Cummings, 1990.

[Gre02] Gregor Engels and Jan Hendrik Hausmann and Reiko Heckel and Stefan Sauer. Testing the Consistency of Dynamic UML Diagrams. In *IDPT*, 2002.

[Hai07] Haitao Dan and Robert M. Hierons and Steve Counsell. *Formal Techniques for Networked and Distributed Systems –FORTE 2007 : 27th IFIP WG 6.1 International Conference Tallinn Estonia*, chapter Thread-Based Analysis of Sequence Diagrams, pages 19–34. Springer Berlin Heidelberg, 2007.

[Hal07] Halvorsen Oddleif and Runde Ragnhild Kobro and Haugen Øystein. *Time Exceptions in Sequence Diagrams*, pages 131–142. Springer Berlin Heidelberg, 2007.

[Har03] Harald Störrle. Semantics of Interactions in UML 2.0. In *HCC*, pages 129–136, 2003.

[Har05] Harald Fecher and Jens Schönborn and Marcel Kyas and de Willem-Paul Roever. 29 New Unclarities in the Semantics of UML 2.0 State Machines. In *Proceedings of the 7th International Conference on Formal Methods and Software Engineering*, ICFEM'05. Springer-Verlag, 2005.

[Has03] Hassan Gomaa and Duminda Wijesekera. Consistency in Multiple-View UML models : A Case Study. In *Workshop on Consistency Problems in UML-based Software Development II*, page 1, 2003.

[Hes06] Hesham Hallal and Sergiy Boroday and Alexandre Petrenko and Andreas Ulrich. A Formal Approach to Property Testing in Causally Consistent Distributed Traces. *Formal Asp. Comput.*, 18(1) :63–83, 2006.

[Hon08] Hong-Viet Luong and Thomas Lambolais and Anne-Lise Courbis. Implementation of the Conformance Relation for Incremental Development of Behavioural Mdels. *In K. Czarnecki et al, editor, MoDELS*, 5301 :356–370, 2008.

[Hui08] Hui Shen and Aliya Virani and Jianwei Niu. Formalize UML2 Sequence Diagrams. *11th IEEE High Assurance Systems Engineering Symposium HASE, Nanjing, China*, pages 437–440, 2008.

[Ing00] Ingolf Heiko Kräuger. Distributed System Design with Message Sequence Charts. *IEEE Trans. Software Eng.*, 2000.

[Iva92] Ivar Jacobson and Magnus Christerson and Patrik Jonsson and Gunnar Övergaard. *Object-Oriented Software Engineering - A Use Case Driven Approach*. Addison-Wesley, 1992.

[J. 96a] J. Davies and J.C.P. Woodcock. Using Z : Specification, Refinement, and Proof. *Prentice-Hall*, 1996.

[J.-96b] J.-R. Abrial. *The B Book*. Cambridge University Press, 1996.

[J. 99a] J. Derrick and EA. Boiten. Non-Atomic Refinement in Z. In *In Wing JM, Woodcock JCP Davies J (eds)*, volume 1708, pages 1477–1496, 1999.

[J. 99b] J. Lilius and I. P. Paltor. Formalising UML State Machines for Model Checking. *UML99-The Unified Modeling Language*, 1999.

[J. 99c] J. Lilius and I. P. Paltor. The Semantics of UML State Machines. *Turku Centre for Computer Science*, 1999.

[J. 99d] J. M. Wing and J. Woodcock and J. Davies, editor. *FM'99 - Formal Methods*, volume 1708 of *LNCS*. Springer, 1999.

[J. 99e] J. Tretmans. Testing Concurrent Systems : A Formal Approach. *In : Baeten J.C.M. Mauw S. (eds.) CONCUR 1999. LNCS vol. 1664. Springer Heidelberg*, 1999.

[J. 01] J. Derrick and E. Boiten. Refinement in Z and Object-Z. 2001.

[J. 05] J. Arjan Mooij and Nicolae Goga and Judi Romijn. Non-local Choice and Beyond : Intricacies of MSC Choice Nodes. In *Fundamental Approaches to Software Engineering, 8th International Conference, FASE 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software ETAPS, Edinburgh, UK, Proceedings*, pages 273–288, 2005.

[J.-07a] J.-R. Abrial. *The Event-B Modelling Notation*, 2007. version 1.5.

[J.-07b] J.-R. Abrial and S. Hallerstede. Refinement, Decomposition, and Instantiation of Discrete Models : Application to Event-B. *Fundamenta Informaticae*, pages 1–28, 2007.

[J.-10] J.-R. Abrial. *Modeling in Event-B : System and Software Engineering.* Cambridge University Press, 2010.

[J.A85] J.A. Bergstra and J.W. Klop. Algebra of Communicating Processes with Abstraction. *Theoretical Computer Science*, pages 77–121, 1985.

[Jac83] M.A. Jackson. System Development. 1983.

[Jac14] Jaco Jacobs and C. Andrew Simpson. On a Process Algebraic Representation of Sequence Diagrams. In *Software Engineering and Formal Methods (SEFM)*, pages 71–85, 2014.

[Jan02] Janette Cardoso and Christophe Sibertin-Blanc. An Operational Semantics for UML Interaction : Sequencing of Actions and Local Control. *European Journal of Automatised Systems, APII-JESA*, 36(7) :1015–1028, 2002.

[Jea02] Jean-Paul Bodeveix and Thierry Millan and Christian Percebois and Christophe Le Camus and Pierre Bazex and Louis Feraud and Ralph Sobek. Extending OCL for Verifying UML Models Consistency. In *Model Engineering, Concepts and Tools*, page 75, 2002.

[Jen10] Jens Bendisposto and Fabian Fritz and Michael Leuschel. Developing Camille, a Text Editor for Rodin. *Workshop on Tool Building in Formal Methods In Proc.*, 2010.

[Joh02] John Derrick and David Akehurst and Eerke Boiten. A Framework for UML Consistency. In *Kuzniarz et al.*, pages 30–45, 2002.

[JR07] M. Fernandes S. Tjell J. B. Jorgensen and O. Ribeiro. Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net. In *6th Int.Wsh. on Scenarios and State Machines*. IEEE Computer Society, 2007.

[Jul06] Juliana Küster Filipe. Modelling Concurrent Interactions. volume 351, pages 203–220, 2006.

[Kev99] Kevin Lano and Andy Evans. Rigorous Development in UML. In *Fundamental Approaches to Software Engineering, Second Internationsl Conference, FASE'99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings*, pages 129–144, 1999.

[Kev00] Kevin Lano and Juan Bicarregui and Andy Evans. Structured Axiomatic Semantics for UML Models. In *Rigorous Object-Oriented Methods, ROOM, York, UK, 17 January 2000*, 2000.

[Kev09] Kevin Lano Sun Meng and Luis S. Barbosa. Co-Algebraic Semantic Framework for Reasoning about Interaction Designs. 2009.

[Kha05] Khadija el Miloudi and Younes el Amrani and Aziz Ettouhami. Using Z Formal Specification for Ensuring Consistency in Multi View Modeling. *Journal of Theoretical and Applied Information Technology 30th. Vol. 57 No.3*, 57(3), 2005.

[L. 92] L. Aceto. Action Refinement in Process Algebras. In *CUP*, 1992.

[Lam02] Leslie Lamport. *Specifying Systems : The TLA+ Language and Tools for Hardware and Software Engineers.* 2002.

[LB03] M. Leuschel and M. Butler. ProB : A Model Checker for B. In Araki Keijiro, Gnesi Stefania, and Mandrioli Dino, editors, *FME 2003 : Formal Methods*, LNCS 2805, pages 855–874. Springer-Verlag, 2003.

[Li 10] Li Dan and Li Danning. An Approach to Formalize UML Sequence Diagrams in CSP. In *3rd International Conference on Computer and Electrical Engineering (ICCEE)*, volume 53, 2010.

[L.M98] L.Mussat and J.-R. Abrial. Introducing Dynamic Constraints in B. In Verlag, editor, *Proceedings of the second International B conference*, volume 1393. Springer, 1998.

[LP02] R. Laleau and F. Polack. Coming and Going from UML to B : A Proposal to Support Traceability in Rigorous IS Development. In *2nd International Conference of B and Z Users*, number 2272, pages 517–534. Springer, 2002.

[Lu 11] Lu Lunjin and Kim Dae-Kyoo. Required Behavior of Sequence Diagrams : Semantics and Refinement. *In 16th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 127–136, 2011.

[M. 87] M. Main. Trace Failure and Testing Equivalences for Communicating Processes. volume 16, pages 383–400, 1987.

[M. 88a] M. Abadi and L. Lamport. The Existence of Refinement Mappings. In *Logic in Computer Science LICS Proceedings of the Third Annual Symposium*, pages 165–175, 1988.

[M. 88b] M. Hennessy. Consistency Problems in UML-Based Software Development. *Technical Report INF-86-13 Dept. of Informatics Twente University of Technology*, 1988.

[M. 06] M. S. Lund and K. Stølen. A Fully General Operational Semantics for UML 2.0 Sequence Diagrams with Potential and Mandatory Choice. *Lecture Notes in Computer Science*, (4085) :380–395, 2006.

[M. 08] M. Leuschel and M. Butler. ProB : An Automated Analysis Toolset for the B Method. *Journal Software Tools for Technology Transfer*, pages 185–203, 2008.

[MA70] L. Mussat and J.-R. Abrial. A Formal Notion of Simulation Between Programs. In *Memo 14, Computers and Logic Research Croup*, 1970.

[Mar04] Maria Victoria Cengarle and Alexander Knapp. UML 2.0 Interactions : Semantics and Refinement. *Technische Universitat München*, pages 85–99, 2004.

[Mar05] Maria Victoria Cengarle and Peter Graubmann and Stefan Wagner and Technische Universität München. Semantics of UML 2.0 Interactions with Variabilities, 2005.

[Mat11] Matthias Schmalz. Term Rewriting in Logics of Partial Functions. *volume 6991 of Lecture Notes in Computer Science Springer Berlin Heidelberg*, pages 633–650, 2011.

[M.B88] M.B. Josephs. A State-Based Approach to Communicating Processes. volume 3, pages 9–18, 1988.

[Mic00] Michael Von der Beeck. Behaviour Specifications :Equivalence And Refinement. 2000.

[Mic05] Michael Leuschel and Michael Butler. Automatic Refinement Checking for B. In Lau Kung-Kiu and Banach Richard, editor, *Formal Methods and Software Engineering*, volume 3785 of *Lecture Notes in Computer Science*, pages 345–359. Springer Berlin / Heidelberg, 2005.

[Mit03] Mitsutaka Okazaki and Toshiaki Aoki and Takuya Katayama. Formalizing Sequence Diagrams and State Machines Using Cncurrent Regular Expression. *Proc. Int. Workshop on Scenarios and State Machines : Models Algorithms and Tools*, pages 74–79, 2003.

[MSD] Describe Control Flow with Fragments on UML Sequence Diagrams. `https://msdn.microsoft.com/en-us/library/dd465153.aspx`.

[N. 71] N. Wirth. Program Development by Stepwise Refinement. In *Commun ACM*, pages 221–227, 1971.

[Nab15] Nabil Messaoudi and Allaoua Chaoui and Mohamed Betta. An Operational Semantics for UML2 Sequence Diagrams Supported by Model Transformations. *The 10th International Conference on Future Networks and Communications (FNC)*, pages 604–611, 2015.

[O. 05] O. Tahir and C. Sibertin-Blanc and J. Cardoso. A Causality-Based Semantics for UML Sequence Diagrams. In *23rd IASTED International Conference on Software Engineering*, pages 106–111. Acta Press, 2005.

[Obj09] Object Management Group. OMG Unified Modeling Language (OMG UML) Superstructure Version 2.2, 2009.

[Obj15] Object Management Group. OMG Unified Modeling Language (OMG UML), Superstructure Version 2.2. *Object Management Group Standard*, 2015.

[Ohl06] Andrè Ohlhoff. *Consistent Refinement of Sequence Diagrams in the UML2.0.* PhD thesis, Christian-Albrechts-Universitat zu Kiel Department of Computer Science Real-time Embedded System Group, 2006.

[OMG13] OMG. OMG Meta Object Facility (MOF) Core Specification Version 2.4.1, 2013.

[Øy03] Øystein Haugen and Ketil Stølen. STAIRS - Steps to Analyze Interactions with Refinement Semantics. In *«UML» 2003 - The Unified Modeling Language, Modeling Languages and Applications, 6th International Conference, San Francisco, CA, USA*, pages 388–402, 2003.

[Øy05] Øystein Haugen and Knut Eilif Husa and Ragnhild Kobro Runde and STAIRS. Towards Formal Design with Sequence Diagrams. In *Software and System Modeling*, volume 4, pages 355–357. John Wiley & Sons Inc., 2005.

[Phi95] Philippe Kruchten. The 4+1 View Model of Architecture. *IEEE Software*, 12(6) :42–50, 1995.

[R. 82] R. Milner. *A Calculus of Communicating Systems.* Springer-Verlag New York Inc., 1982.

[R. 89] R. Milner. *Communication and Concurrency.* Prentice Hall London UK, 1989.

[R. 93] R. van Glabbeek. The linear Time-Branching Time Spectrum. Lecture Notes in Computer Science. Springer-Verlag, 1993.

[R. 99] R. Milner. Communicating and Mobile Systems : The Pi Calculus. *Cambridge University Press, Cambridge*, 1999.

[R. 05] R. Grosu and SA. Smolka. Safety-Liveness Semantics for UML 2.0 Sequence Diagrams. In *5th Int. Conf. on Application of Concurrency to System Design*, pages 6–14, 2005.

[Rag03a] Ragnhild Van Der Straeten and Tom Mens and Jocelyn Simmonds and and Viviane Jonckers. Using Description Logic to Maintain Consistency Between UML Models. In *Commun ACM*, pages 326–340, 2003.

[Rag03b] Ragnhild Van Der Straeten and Tom Mens and Jocelyn Simmonds and Viviane Jonckers. Using Description Logic to Maintain Consistency Between UML Models. pages 326–340, 2003.

[Rag04] Ragnhild Van Der Straeten and Viviane Jonckers and Tom Mens. Supporting Model Refactorings through Behaviour Inheritance Consistencies. In *The Unified Modelling Language*, pages 305–319, 2004.

[Rag05] Ragnhild Kobro Runde and Øystein Haugen and Ketil StØlen. How to Transform UML Neg into a Useful Construct. In *Proc. Norsk Informatik konferanse. Tapir*, pages 55–66, 2005.

[Rag07] Ragnhild Kobro Runde. *STAIRS - Understanding and Developing Specifications Expressed as UML Interaction Diagrams.* PhD thesis, 2007.

[Raj96] Rajeev Alur and Gerard J. Holzmann and Doron Peled. An Analyzer for Message Sequence Charts. In *Software concepts and tools*, pages 304–313, 1996.

[Ral78] Ralph-Johan Back. On the Correctness of Refinement Steps in Program Development. In *Ph.D. thesis, Department of Computer Science University of Helsinki*, 1978.

[Ral94] Ralph-Johan Back and Kaisa Sere. From Action Systems to Modular Systems. Reports on Computer Science and Mathematics 154, 1994.

[Ran93] Rance Cleaveland and Matthew Hennessy. Testing Equivalence as a Bisimulation Equivalence. *Formal Aspects of Computing*, 5 :1–20, 1993.

[R.J98] R.J. Back and J. von Wright. Refinement Calculus : A Systematic Introduction. *Graduate Texts in Computer Science. Springer-Verlag*, 1998.

[Rob71] Robin Milner. An Algebraic Definition of Simulation Between Programs. Technical report, Stanford, CA, USA, 1971.

[ROD07] RODIN. Rigorous Open Development Environment for Complex Systems, 2007. `http://rodin-b-sharp.sourceforge.net`.

[Rum96] E. James Rumbaugh. To Form a More Perfect Union : Unifying the OMT and Booch Methods. *JOOP*, pages 14–18, 1996.

[S. 94] S. Gerhart and D. Craigen and T. Ralston. Case study : Paris Metro Signaling System. *IEEE Software*, 11(1) :32–28, 1994.

[S. 05] S. W. Lam Vitus and Julian Padget. Consistency Checking of Sequence Diagrams and Statechart Diagrams Using the PI-calculus. In *Proceedings of the 5th International Conference on Integrated Formal Methods*, IFM'05, pages 347–365. Springer-Verlag, 2005.

[Seb01] Sebastian Uchitel and Jeff Kramer and Jeff Magee. Detecting Implied Scenarios in Message Sequence Chart Specifications. In *Proceedings of the 8th European Software Engineering Conference held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 74–82. ACM New York NY USA, 2001.

[Sha11] Shahar Maoz and David Harel and Asaf Kleinbort. A Compiler for Multimodal Scenarios : Transforming LSCs into AspectJ. *ACM Trans. Softw. Eng. Methodol.*, 2011.

[She13] Shen Hui. *A Formal Framework for Analyzing Sequence Diagram.* PhD thesis, 2013.

[Sim04] Simon Pickin and Jean-Marc Jézéquel. Using UML Sequence Diagrams as Basis for a Formal Test Description Language. In *Proc. of Fourth International Conference on Integrated Formal Methods IFM2004*. Springer, 2004.

[Siw10] Siwik Leszek and Lewandowski Krzysztof and Wos Adam and Drezewski Rafal and Kisiel-Dorohinicki Marek. UML2SQL - A Tool for Model-Driven Development of Data Access Layer. In Szczerbicki Edward and Nguyen Ngoc Thanh, editor, *Smart Information and Knowledge Management*, Studies in Computational Intelligence, pages 227–246. Springer, 2010.

[Ste99] Stefania Gnesi and Diego Latella and Mieke Massink. Modular Semantics for a UML Statechart Diagrams kernel and its Extension to Multicharts and Branching time Model-Checking. *Journal of Logic and Algebraic Programming 51*, (1) :43–75, 1999.

[Ste14] Steve Schneider and Helen Treharne and Heike Wehrheim and David M. Williams. Managing LTL Properties in Event-B Refinement. In Elvira Albert and Emil Sekerinski, editors, *Integrated Formal Methods*, pages 221–237. Springer International Publishing, 2014.

[Stu99] Stuart Kent and Andy Evans and Bernhard Rumpe. UML Semantics FAQ. In *Object-Oriented Technology ECOOP'99 Workshop Reader Panels and Posters Lisbon Portugal*, pages 33–56, 1999.

[Sve04] Sven Burmester and Holger Giese and Martin Hirsch and Daniela Schilling. Incremental Design and Formal Verification with UML/RT in the FUJABA Real- Time Tool suite. *SVERTS*, 2004.

[Tim01] Timm Schäfer and Alexander Knapp and Stephan Merz. Model Checking UML State Machines and Collaborations. *Electronic Notes in Theoretical Computer Science 55*, (3) :357–369, 2001.

[Vit09] Vitaly Savicks and Colin Snook and Michael Butler. Animation of UML-B Statemachines. *Rapport technique University of Southampton*, 2009.

[Zol11] Zoltán Micskei and Hélène Waeselynck. The Many Meanings of UML2.0 Sequence Diagrams : a Survey. *Software & Systems Modeling*, pages 489–514, 2011.