N° d'ordre: 3496

THÈSE

présentée

devant l'Université de Rennes 1

pour obtenir

le grade de : DOCTEUR DE L'UNIVERSITÉ DE RENNES 1
Mention INFORMATIQUE

par

Jacques KLEIN

Équipe d'accueil : Equipe Triskell IRISA
École Doctorale : Matisse
Composante universitaire : IFSIC

Titre de la thèse :

Aspects Comportementaux et Tissage

À soutenir le X décembre 2006 devant la commission d'examen

M. :	Françoise	ANDRÉ	Président
MM. :	Laurence	DUCHIEN	Rapporteurs
	Anca	MUSCHOLL	
MM. :	Nicolas	GUELFI	Examinateurs
	Loïc	HÉLOUËT	
	Jean-Marc	JÉZÉQUEL	

A Sophie,

Diviser pour mieux régner
par Machiavel

Remerciements

%%%%%%%%%%%%% **DRAFT** %%%%%%%%%%%%%

Je remercie Françoise ANDRE, Professeur à l'Université de Rennes 1, qui me fait l'honneur de présider ce jury.

Je remercie Laurence DUCHIEN, Professeur à l'Université de Lille 1, et Anca MUSCHOLL, Professeur à l'Université Paris 7, d'avoir bien voulu accepter la charge de rapporteur.

Je remercie Nicolas GUELFI, Professeur à l'Université du Luxembourg, d'avoir bien voulu juger ce travail.

Je remercie Loïc HELOUET, Chargé de Recherche INRIA, d'avoir co-encadré ma thèse.

Je remercie enfin Jean-Marc JEZEQUEL, Professeur à l'Université de Rennes 1, qui a dirigé ma thèse.

%%%%%%%%%%%%% **DRAFT** %%%%%%%%%%%%%

Table des matières

Introduction

Le génie logiciel propose des solutions pratiques, fondées sur des connaissances scientifiques, pour produire et maintenir des logiciels avec des contraintes de coût, de temps et de qualité. Il est admis que la complexité d'un logiciel augmente fortement en fonction de sa taille. Cependant, d'un côté, la taille d'un logiciel est en moyenne multipliée par dix tous les dix ans, et d'un autre côté, les pressions économiques imposent un rendement maximum lors du développement d'un logiciel (diminution de la durée de développement, diminution des coûts, maintenance accrue, etc.).

Face à ces problèmes, des approches nouvelles apparaissent, en particulier celles fondées sur les concepts du développement logiciel orienté aspect (AOSD, Aspect-Oriented Software Development) identifié par le MIT comme une des dix technologies émergentes de la décennie pouvant dans un délai proche avoir un profond impact dans notre quotidien, notre travail et notre économie. D'un point de vue général, la communauté AOSD préconise de suivre des idées simples connues de longue date telles que le principe romain "divide et imperia[1]" : il est plus facile de gérer un grand système s'il est découpé en petits morceaux, et en particulier si les solutions relatives à des sous-systèmes peuvent être combinées pour former une solution correspondante au système global.

Plus précisément, la communauté AOSD préconise de traiter des préoccupations transversales (tels que la sécurité, la synchronisation, la concurrence, la persistance, le temps de réponse, ...) en fournissant des moyens systématiques permettant leurs identifications, séparations, représentations et compositions. Chacune de ces préoccupations, appelées aspect, est encapsulée dans un seul module. Une fois différents aspects spécifiés, ils peuvent être assemblés et fusionnés dans une application de base pour former une application globale. Ce mécanisme d'intégration est appelé tissage. Il en résulte, entre autres, une meilleure modularité et une diminution des coûts de développement, de maintenance et d'évolution. La modularité des préoccupations transversales a été popularisée par le langage de programmation Aspect-J, mais récemment, un nombre grandissant de travaux se focalisent sur le tissage dans des phases amont du cycle de vie d'un logiciel, et en particulier sur le tissage d'aspects à un niveau de modélisation. On parlera alors de tissage d'aspects dans un modèle de base.

Cependant, il n'existe pas de tisseur automatique à un niveau de modélisation qui aurait une utilité certaine. En effet, outre le fait de modéliser indépendamment chaque

[1] repris par Machiavel sous l'expression "diviser pour mieux régner"

préoccupation, ce qui facilite leur maintenabilité ou encore leur réutilisabilité, un moyen de tisser des préoccupations transverses à un niveau de modélisation permettrait de constater la réelle nature des interactions entre les diverses préoccupations avant une phase d'implémentation, où le coût de correction d'éventuelles erreurs est plus élevé. Tisser des préoccupations à un niveau de modélisation permettrait alors d'avoir une vision précoce d'un système dans sa globalité, pour ainsi pouvoir le valider le plus tôt possible et éviter la propagation d'erreurs à travers l'ensemble du cycle de développement logiciel. Un autre argument en faveur de l'utilisation d'un tisseur de modèles est la possibilité d'utiliser un développement par aspects même si une plateforme cible ne supporte pas des mécanismes de tissages d'aspects (par exemple, les plateformes de systèmes embarqués), car les aspects pourraient être tissés au niveau des modèles, avant la prise en compte d'une plateforme cible.

Un processus de tissage est généralement décomposé en deux phases : une phase de détection où une partie d'un aspect, appelée *expression de coupe* (*pointcut* en anglais[2]), est utilisée comme prédicat pour identifier toutes les parties du modèle de base où l'aspect doit être tissé (ces parties sont appelées *points de jonction*) ; et une phase de composition où une deuxième partie d'un aspect, appelée *advice* représentant un comportement désiré, est composée ou fusionnée avec le modèle de base aux endroits précédemment détectés.

Travaux de thèse

Dans cette thèse, nous nous intéressons au tissage d'aspects au niveau de l'analyse et de la conception des systèmes. En particulier, nous traitons du tissage d'aspects comportementaux exprimés à l'aide de scénarios dans des modèles de base également exprimés à l'aide de scénarios. Les scénarios décrivent des comportements désirés ou existants d'un système, et plus précisément les échanges de messages entre les entités composant ce système. Ils sont utilisés durant le cycle de développement de logiciel (par exemple, pour raffiner les cas d'utilisations) ou pour décrire le comportement de systèmes distribués, car ils sont graphiques et facilement compréhensibles.

Problèmatiques abordées

L'utilisation d'aspects comportementaux à un niveau de modélisation a pour objectif, en autres, de permettre de modifier le comportement d'un modèle de base, et de permettre d'exprimer des expressions de coupe "dynamiques" pour déterminer où un aspect doit être tissé. Dans ce contexte, le problème du tissage d'aspects comportementaux à un niveau de modélisation tient en trois points :

- Premièrement, il est préférable de décrire un aspect en utilisant des langages de modélisation dynamiques, plutôt que d'utiliser des modèles statiques en les complexifiant pour faire apparaître du dynamisme.

[2]La version française des termes *pointcut*, *join point* et *advice* est empruntée de la thèse de Barais [Bar05]

- Deuxièmement, puisque nous devons tisser un aspect dynamique à un niveau de modélisation, il est nécessaire d'identifier statiquement dans le modèle de base les endroits où l'aspect doit être tissé. Autrement dit, il est nécessaire de déterminer statiquement les points de jonction. Alors que cette recherche peut être accomplie facilement en ne considérant que la syntaxe du modèle utilisé, le problème devient plus difficile si la sémantique du modèle utilisé est prise en compte. Ce mécanisme d'identification des points de jonction, aussi appelé langage de définition des expressions de coupe, joue un rôle crucial dans l'applicabilité d'une méthodologie orientée-aspect. En effet, comme un langage d'expressions de coupe permet de spécifier où un aspect coupe le programme ou le modèle de base, plus ce langage est puissant[3] et précis, plus il est facile de séparer une préoccupation transversale d'un modèle de base et d'indiquer de manière non ambiguë où elle doit être tissée (sans pour autant l'indiquer explicitement). Selon Kiczales [Kic03], le langage de définition d'expressions de coupe a probablement le rôle le plus important dans le succès d'une technologie orientée-aspects. Cependant Kiczales souligne que la plupart des solutions proposées jusqu'ici sont liées de manière excessive à la syntaxe des programmes ou des modèles manipulés.
- Le troisième problème concerne la composition de l'advice (le comportement désiré) avec le modèle de base. Le modèle utilisé étant dynamique, lors de cette composition il est nécessaire de tenir compte de la sémantique du modèle utilisé et de fournir des opérateurs de compositions définis formellement pour obtenir des résultats cohérents.

Contributions

Cette thèse répond aux trois points soulevés par la problématique pour des modèles de scénarios :

- Premièrement, nous définissons un aspect comme une paire de scénarios finis, où le premier scénario est utilisé pour spécifier une expression de coupe, c'est-à-dire le comportement à détecter, et où le deuxième scénario est utilisé pour spécifier l'advice, c'est-à-dire le comportement voulu au niveau des parties préalablement détectées (points de jonction).
- Deuxièmement, nous proposons un langage d'expressions de coupe qui tient compte de la sémantique du langage de scénarios utilisé. Pour cela, dans un premier temps, nous proposons diverses sémantiques de points de jonction facilitant plus ou moins le tissage d'aspects multiples, ainsi qu'un moyen d'ordonner des points de jonction successifs. Dans un deuxième temps, nous proposons des algorithmes de détection de points de jonction dans des scénarios finis. Finalement, nous proposons des algorithmes de détection dans des scénarios infinis. Ces algorithmes prennent en compte l'ordre partiel induit par des scénarios, mais ils prennent également en compte l'ensemble potentiellement infini de comportements généré par des scénarios de haut niveau grâce à des techniques de dépliage de boucle ou encore de

[3]Un compromis entre puissance et faisabilité est tout de même nécessaire, car si le langage est trop puissant, beaucoup de problèmes deviennent indécidables

permutation. De cette manière, la détection proposée est réellement fondée sur la sémantique du modèle utilisé. Ces travaux ont été en partie présentés dans [KHJ, KF06].

- Le troisième problème est résolu en définissant des opérateurs de composition pour scénarios. Jusqu'ici, la composition de scénarios était limitée à une composition parallèle ou séquentielle, à des itérations ou des choix. Cependant, quand deux scénarios décrivent différents points de vue d'un même comportement, il est nécessaire d'introduire un opérateur de composition qui fusionne les deux scénarios pour produire un résultat contenant les deux opérandes sans créer de copie des éléments similaires. Cet opérateur ne peut pas être exprimé à l'aide de simples compositions séquentielles ou parallèles. Nous proposons alors un opérateur de fusion de scénarios appelé *somme amalgamée* pour composer des scénarios finis, et un opérateur appelé *produit fibré* pour composer des scénarios infinis. Ces travaux ont été en partie présentés dans [KCH04, KF06].

La résolution des problèmes soulevés nous a permis de développer un tisseur d'aspects pour scénarios dans la plateforme Kermeta développée dans l'équipe Triskell, et dont l'utilité est montrée sur des études de cas.

Organisation du document

Après ce chapitre d'introduction, nous présentons dans le chapitre 1, le contexte et le langage de scénarios utilisés dans cette thèse. Nous définition également de manière plus précise la notion d'aspect comportemental. Dans le chapitre 2, nous présentons notre langage d'expressions de coupe et des algorithmes de détection de points de jonction dans des comportements finis. Le chapitre 3 est consacré à la présentation d'un mécanisme de détection dans des comportements infinis. Le chapitre 4 présente de nouveaux opérateurs de compositions. Le Chapitre 5 décrit la mise en oeuvre dans l'environnement Kermeta du tisseur de scénarios développé dans cette thèse, ainsi que plusieurs cas d'étude. Finalement, le chapitre 6 conclut cette thèse.

Chapitre 1

Modélisation Orientée Aspect et Scénarios

L'objectif de ce chapitre est double. Premièrement, nous expliquons, à travers les travaux existants, pourquoi nous nous sommes intéressés au tissage de scénarios, et nous montrons les points faibles ou les manques que nous proposons de combler. Le deuxième objectif est de présenter notre concept d'aspects comportementaux, et les grandes lignes du mécanisme de tissage que nous proposons. Pour cela, nous divisons ce chapitre en quatre sections :

La première est consacrée à une présentation du contexte général dans lequel se placent nos travaux, c'est-à-dire l'Ingénierie Dirigée par les Modèles (IDM), et en particulier la Modélisation Orientée-Aspects (MOA) qui peut être vue comme une approche spécifique de l'IDM dans laquelle chaque préoccupation est modélisée séparément.

La deuxième section présente les travaux existants portant sur la modélisation orientée-aspect. Nous remarquerons que la plupart des travaux se focalisent sur la modélisation de programmes écrits avec des langages de programmation orientés-aspect, et qu'ils ne proposent pas de véritable tisseur d'aspects, défini formellement, pour les modèles. Nous identifions peu d'approches dont le but est de véritablement tisser des modèles, mais ces approches ne considèrent que des diagrammes structuraux, tels que les diagrammes de classe. Bien qu'utile, ce type de tissage n'est pas suffisant, et surtout, il n'est pas adapté au tissage d'aspects liés aux comportements d'un système. Ce constat a motivé notre choix de proposer un tisseur d'aspects comportementaux exprimés par des scénarios, et plus précisément par des Message Sequence Charts (MSCs).

La troisième section présente les MSCs et quelques définitions et problèmes intéressants s'y rapportant.

Finalement, la quatrième section présente notre concept d'aspects comportementaux, ainsi que les grandes lignes de notre mécanisme de tissage pour scénarios.

1.1 Ingénierie Dirigée par les Modèles et Modélisation Orientée Aspect

Les deux sous-sections suivantes ont pour but de présenter le contexte dans lequel se placent les travaux de cette thèse. La sous-section portant sur la présentation de l'ingénierie dirigée par les modèles est inspirée par le chapitre 2 de la thèse de Fleurey [Fle06]

1.1.1 Ingénierie Dirigée par les Modèles

L'Ingénierie dirigée par les modèles (IDM) [EFB+05] propose des solutions permettant de répondre aux exigences de plus en plus fortes du développement de logiciel qui imposent de produire et de maintenir des logiciels avec des contraintes de coût, de temps et de qualité, alors que la taille et la complexité des logiciels ne cessent d'augmenter. L'IDM est une forme d'ingénierie générative, par laquelle tout ou une partie d'un logiciel est généré à partir de modèles. Les idées de base de cette approche sont voisines de celles de nombreuses autres approches du génie logiciel, comme la programmation générative, les langages spécifiques de domaines (DSL pour *domain-specific language*) [vDKV00, CM98], le MIC (Model Integrated Computing) [SKB+95, SK97], les usines à logiciels (Software Factories) [GSC+04], etc.

Dans cette nouvelle perspective, les modèles occupent une place de premier plan parmi les artefacts de développement des systèmes, et doivent en contrepartie être suffisamment précis afin de pouvoir être interprétés ou transformés par des machines. Le processus de développement des systèmes peut alors être vu comme un ensemble de transformations de modèles partiellement ordonnés, chaque transformation prenant des modèles en entrée et produisant des modèles en sortie, jusqu'à obtention d'artefacts exécutables. Le plus souvent, une transformation permet de raffiner un modèle d'entrée. Autrement dit, un modèle de sortie d'une transformation est, en général, moins abstrait ou plus raffiné que le modèle d'entrée. Nous pouvons donc dire que les transformations suivent un axe "vertical", où chaque transformation fait passer un modèle d'un niveau d'abstraction donné, à un modèle de niveau d'abstraction moindre.

Pour donner aux modèles cette dimension opérationnelle, il est essentiel de spécifier leur sémantique, et donc aussi de décrire de manière précise les langages utilisés pour les représenter. On parle alors de métamodélisation.

L'intérêt pour l'Ingénierie Dirigée par les Modèles a été fortement amplifié, en novembre 2000, lorsque l'OMG (*Object Management Group*) a rendu publique son initiative MDA [StOs00] qui vise à la définition d'un cadre normatif pour l'IDM. Il existe cependant bien d'autres alternatives technologiques aux normes de l'OMG, telles que Ecore dans la sphère Eclipse, mais aussi les grammaires, les schémas de bases de données, ou encore les schémas XML. Finalement, l'IDM dépasse largement le MDA, pour se positionner plutôt à la confluence de différentes disciplines.

1.1.2 Modélisation Orientée-Aspect

Le terme *préoccupation transversale*, ou aspects (par exemple, sécurité, temps de réponse, persistance, ...) est associé aux fonctionnalités d'un logiciel qui ne peuvent pas être efficacement isolées dans un seul module en utilisant les techniques actuelles du développement logiciel, telles que les approches orientées-object. Décomposer un système en préoccupations n'ayant qu'un intérêt limité si ces préoccupations ne peuvent être composées pour former un système complet, la communauté orientée-aspect propose également des moyens de composition qui permettent ce que l'on appelle le *tissage des préoccupations*, ou le *tissage d'aspects*.

Le développement logiciel orienté-aspect : L'idée de la séparation des préoccupations, c'est-à-dire l'identification de différentes préoccupations dans le développement logiciel et leur séparation en les encapsulant dans des modules appropriés ou dans des parties spécifiques d'un logiciel, peut être associée à Dijkstra [Dij76] et Parnas [Par72]. Le développement logiciel orienté-aspect (AOSD pour Aspect-Oriented Software Development) adopte ces idées et a pour objectif supplémentaire de fournir un nouveau moyen d'encapsulation de préoccupations transversales, appelées *aspects*. Cette idée apparaît aujourd'hui comme une manière judicieuse de compléter la notion de modules disponible dans la plupart des langages de programmation. La séparation de préoccupations transverses permet au concepteur de logiciel d'avoir un meilleur contrôle sur les variations (dans le contexte des lignes de produits par exemple) et les évolutions du logiciel.

L'AOSD, également appelé par le passé *séparation avancée des préoccupations* (Advanced Separation of Concerns), est un domaine de recherche relativement jeune, mais à l'activité croissante. L'AOSD représente la convergence de différentes approches de séparation de préoccupations, telles que la Programmation Adaptative [Lie96], les "Composition Filters" [ABV92], la programmation orientée-sujet [OKK+96, HO93] ou encore la programmation orientée-aspect[KLM+97].

La Modélisation Orientée-Aspect : d'un point de vue du développement logiciel, l'AOSD a émergé à un niveau de programmation, avec notamment Aspect-J [KHH+01] qui a joué un rôle déterminant dans l'émancipation de cette approche. Pourtant, à travers l'importance croissante de l'IDM, le paradigme orienté-aspect ne s'est plus restreint au niveau de la programmation, et il s'étend maintenant aux phases amonts du développement logiciel, par exemple au niveau de la conception, de l'analyse [Cla01] ou encore de l'étude des exigences [AWK04, WA04, RMA03, JN04] d'un système. Dans ce contexte, la modélisation orientée-aspect (MOA) étend le processus de développement de l'IDM (qui propose des transformations verticales), en découpant le modèle d'un système en plusieurs préoccupations à un même niveau d'abstraction, et en les composant à travers un processus de tissage qui peut être vu comme une transformation "horizontale" de modèles. Outre le fait de séparer les préoccupations transverses tôt dans le cycle de développement, couplé à des mécanismes automatiques de tissage d'aspects, la modélisation orientée-aspect pourrait permettre de cibler des plates-formes non orientées-aspect (par exemple java sans extension ou système temps réel embarqué) ou bien faciliter la mise en place de techniques de validation comme la simulation ou la

génération de tests.

Dans le langage de programmation orienté-aspect Aspect-J [KHH+01], un processus de tissage est décomposé en deux phases. Une phase de détection où une partie d'un aspect (appelée *expression de coupe*) est utilisée comme prédicat sur un programme dit de base pour déterminer toutes les zones (appelées *points de jonction*) où l'aspect doit être tissé. Une deuxième phase consiste à composer une deuxième partie d'un aspect (appelée *advice*) avec le programme de base aux endroits préalablement détectés, c'est-à-dire au niveau des points de jonction.

Afin de rendre utilisable la séparation des préoccupations au cours des phases de modélisation, les notions d'aspect, de point de coupure et de tissage popularisées par Aspect-J doivent être précisément définies pour le formalisme de modélisation utilisé. Le mécanisme d'identification des points de jonction, aussi appelé langage de définition des expressions de coupe (ce qui correspond à la première phase d'un processus de tissage), joue un rôle crucial dans l'applicabilité d'une méthodologie orientée-aspect. En effet, comme un langage d'expression de coupe permet de spécifier où un aspect coupe le programme ou le modèle de base, plus ce langage est puissant et précis, plus il est facile de séparer une préoccupation transversale d'un modèle de base et d'indiquer de manière non ambiguë où elle doit être tissée (sans pour autant l'indiquer explicitement). Selon Kiczales [Kic03], le langage de définition des expressions de coupe a probablement le rôle le plus important dans le succès d'une technologie orientée-aspects. Cependant, Kiczales souligne que la plupart des solutions proposées jusqu'ici sont liées de manière excessive à la syntaxe des programmes ou des modèles manipulés. Les chapitres 2 et 3 de cette thèse sont consacrés à la définition d'un langage d'expression de coupe tenant compte de la sémantique du langage de modélisation utilisé.

Quelques définitions relatives à la MOA

Avant de présenter, dans la section suivante, différentes approches de modélisation par aspects, nous allons présenter quelques définitions relatives aux principaux concepts l'approche orientée-aspect.

Préoccupation : Dans [KvdBC05], une préoccupation est définie comme un intérêt relatif au développement d'un système, ayant un rôle critique ou important à un moment donné du développement. Une préoccupation est alors soit une préoccupation non transversale, également appelée préoccupation de base. Soit une préoccupation transversale, également appelée préoccupation d'aspect ou simplement aspect.

Préoccupation de base ou Base : Une base est une unité de modularisation qui peut être représentée dans un module de manière isolée en respectant une *décomposition dominante* [TOHJ99]. Elle représente donc une préoccupation non transversale qui peut être capturée de manière efficace avec des approches traditionnelles telles que l'approche orientée object.

Préoccupation d'aspect ou Aspect : Un aspect est une unité de modularisation qui est entremêlée avec les autres préoccupations, qu'elles soient de base ou d'aspect.

Tissage : Le tissage d'un aspect est le processus de composition de l'aspect avec les autres préoccupations. Le tissage de tous les aspects relatifs à un système produit

le *modèle tissé* du système. Après tissage, les aspects cessent d'exister séparément. On peut distinguer deux types de tissage : le tissage statique (avant l'exécution) et le tissage dynamique (durant l'exécution). La figure 1.1 donne un exemple de tissage statique. L'automate de gauche est transformé en l'automate de droite où chaque transition a ont été remplacée par une transition c. Un tissage dynamique serait plutôt exprimé sous la forme : "lors de l'exécution de l'automate de gauche, à chaque fois que la transition a est tirée, effectuer une action c".

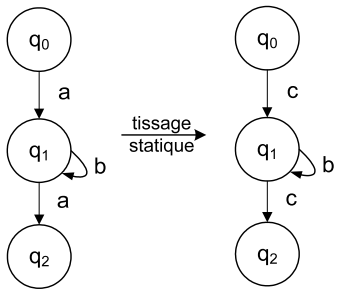


FIG. 1.1 – Illustration d'un tissage statique

Expression de coupe : Prédicat sur une préoccupation de base permettant de déterminer toutes les zones (appelées *points de jonction*) où l'aspect doit être tissé.

Point de jonction : Un point de jonction représente une zone où un aspect est entremêlé avec une autre préoccupation.

Modèle de Points de jonction : Le modèle de points de jonction est composé de tous les éléments d'un langage donné qui peuvent former un point de jonction.

Advice : Modifications apportées par un aspect dans les préoccupations de base au niveau des points de jonction.

Dans la section suivante, nous allons présenter les principales techniques existantes de modélisation orientée-aspect, qui même si elles proposent des concepts intéressants, ne proposent pas véritablement de tisseur d'aspects exprimés à l'aide d'un modèle donné.

1.2 Différentes Approches de Modélisation Orientées-Aspect

Les travaux présentés dans la section suivante, et les travaux présentés dans cette thèse, se placent plutôt juste après la phase d'étude des exigences, c'est-à-dire au niveau de l'analyse et la conception des systèmes. Pour cette raison, nous ne présenterons pas les travaux liés au domaine particulier des "aspects précoces " [ear] (Early Aspect en anglais) qui s'intéressent à la problématique des aspects au niveau de l'étude des exigences.

Il existe un grand nombre d'approches de modélisation orientées-aspect, comme nous le montre l'étude proposée par le réseau européen d'excellence AOSD-Europe [Chi05]. Nous allons présenter cinq de ces approches qui nous apparaissent les plus représentatives. Nous critiquerons chacune de ces approches en insistant sur les mécanismes de composition et de détection de points de jonction qu'elles proposent. Une synthèse sur l'étude de ces travaux sera également proposée.

1.2.1 Approche de modélisation orientée-aspect de France et al.

Aperçu : L'approche de France et al. [FRGG04, RFLG04] est basée sur UML2.0, et récemment, elle a été améliorée dans [RGF+06, RFG+05] en proposant des mécanismes de composition plus aboutis, mais concernant essentiellement les diagrammes de classe. Les modèles d'aspects (les préoccupations d'aspects modélisées) sont représentés en utilisant des "diagrammes templates"[1]. Plus précisément, ce sont des templates de diagrammes de classe et de diagrammes de communications qui décrivent respectivement les parties structurelles et comportementales des aspects. Pour améliorer la lisibilité de leurs aspects, les auteurs préfèrent fournir une notation des templates différentes de celle proposée par UML2.0. France et al. proposent également une brève description d'un processus de conception de modélisation orientée-aspect.

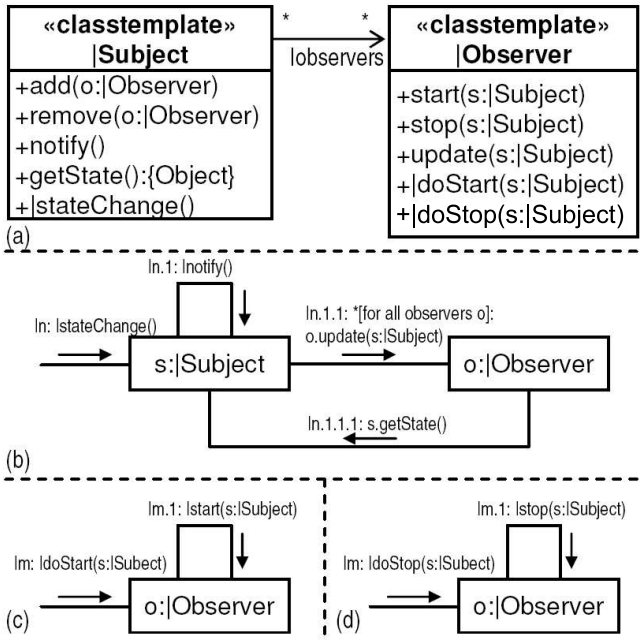


FIG. 1.2 – Le modèle d'aspect d'un patron sujet-observateur en utilisant la notation de France et al.

Composition : France et al. établissent une distinction entre les modèles d'aspect et les modèles de base (qu'ils appellent modèles principaux ou primary model en anglais). Les modèles d'aspect sont modélisés en utilisant des diagrammes templates, qui sont décrits par des paquetages paramétrés (figure 1.2). A partir des diagrammes template, une *association* textuelle (et donc explicite) à une application donnée, permet l'instanciation d'un modèle d'aspect spécifique à un contexte donné (figure 1.3). L'association suivante instancie le modèle d'aspect de la figure 1.2 et permet l'obtention du modèle d'aspect spécifique à un contexte, représenté sur la figure 1.3 :

[1] template diagrams en anglais

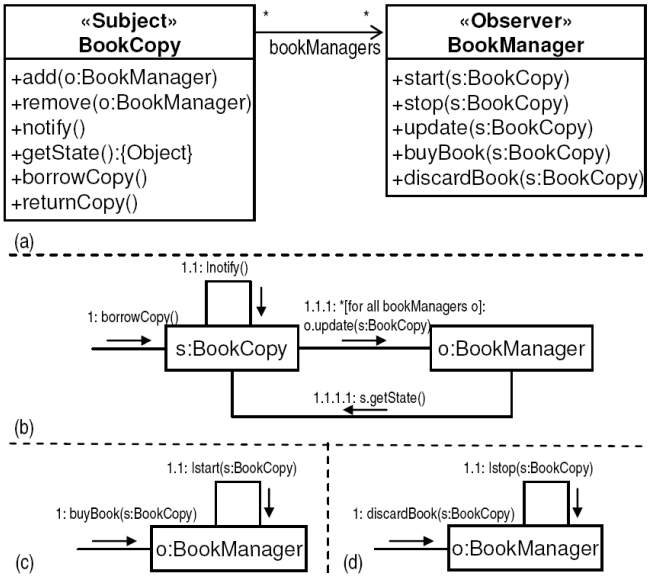


FIG. 1.3 – Un modèle d'aspect spécifique à un contexte donné en utilisation l'approche de France et al.

```
(|Subject,BookCopy);                (|Observer, BookManager);
(|stateChange(), borrowCopy());    (|doStart(s :|Subject),buyBook());
(|stateChange(),returnCopy());     (|doStop(s :|Subject),discardBook());
(|observer,bookManagers);
```

Le modèle d'aspect spécifique à un contexte est ensuite composé avec un modèle de base. Dans [RGF+06], les auteurs introduisent un mécanisme de composition, appelé *signature-based composition*[2], supporté par des *directives de composition*. Malheureusement, le mécanisme de composition n'est pour l'instant appliqué qu'aux diagrammes structurels. Par contre, il est défini en termes de composition de métamodèle et il est implanté dans l'environnement Kermeta. Les directives de composition servent à la résolution de conflits entre éléments (par exemple, lorsque deux classes ont le même nom). Les directives de composition permettent également, entre autres, d'ajouter, de supprimer, de remplacer des éléments d'un modèle. Finalement, ils contiennent des directives telles que "precedes" et "follows" dérivées de la métaclasse *Dependency* d'UML, permettant de spécifier un ordre de composition des modèles d'aspect avec le modèle de base.

Conclusion : L'approche de modélisation orientée-aspect de France et al. est une des approches existantes les plus avancées. Ils proposent des mécanismes de composition évolués qui sont réellement implantés et donc utilisables. Nous pouvons tout de même faire deux critiques majeures à leur approche. Premièrement, ils ne proposent pas de mécanisme de détection de points de jonction. Deuxièmement, les mécanismes

[2]La signature d'un élément est définie par ces propriétés syntaxiques, où une propriété syntaxique est soit un attribut, soit une association définie dans le métamodèle de classe d'UML

de composition sont restreints aux modèles statiques.

1.2.2 Approche de modélisation orientée-aspect de Clarke et al. : Theme

Aperçu : L'approche *Theme* de Clarke et al. [CB05, CW04, CW02, BC04] porte sur le domaine de l'AOSD au niveau des phases d'étude des exigences et d'analyse avec *Theme/Doc*[3], et au niveau de la phase de conception avec *Theme/UML*. Nous allons nous focaliser sur Theme/UML qui est utilisé pour produire une décomposition d'un système en *themes*, un theme permettant l'encapsulation d'une préoccupation. Theme/UML est fondé sur une extension du métamodèle d'UML1.3, et son origine remonte aux travaux de Clarke dans le domaine de l'approche orientée-sujet [CW01]. De manière générale, Theme/UML ne présente pas de restriction portant sur l'utilisation d'un modèle particulier d'UML. Néanmoins, Theme/UML est le plus souvent présenté comme une approche utilisant les diagrammes de classes pour décrire la partie structurelle d'un système, et utilisant les diagrammes de séquences pour décrire le comportement de celui-ci.

Composition : Theme/UML est une approche dite symétrique, c'est-à-dire que Theme n'établit pas de distinction forte entre un modèle de base et un modèle d'aspect. Chaque préoccupation est encapsulée dans un theme, qui est un paquetage UML portant le stéréotype $\ll$ *theme* $\gg$. La figure 1.4 décrit deux themes (Observer et Library). Les themes d'aspect présentent tout de même une distinction par rapport aux themes de base. En effet, un theme d'aspect comporte un mécanisme de template (similaire à celui de France et al.) qui permet l'instanciation de paramètres template. Trois types de compositions sont posssibles : la fusion, le remplacement (pour la composition de deux themes de bases), et l'association (pour la composition d'un theme de base avec un theme d'aspect) [4]. Le tissage des deux themes de la figure 1.4 est représenté dans la figure 1.5. Comme le theme *Observer* est un theme d'aspect, la composition utilise une association. Sur la figure 1.4, nous pouvons voir la description de l'association (*Bind*[...]) qui est liée au template du theme *Observer*. La classe *BookCopy* joue le rôle de *Subject*, les méthodes *borrowCopy*() et *returnCoppy*() jouent le rôle de *_aStateChange*(..), etc.

Conclusion : Theme est une approche dont l'intérêt majeur porte sur la méthodologie qu'elle propose. A l'opposé, bien que Theme utilise des *themes* qui comportent une partie structurelle et une partie comportementale, les inconvénients de themes sont, premièrement qu'il n'existe pas d'outils et de formalisations des mécanismes de composition. Deuxièmement, Theme ne propose pas de mécanisme de détection de points jonction.

[3] Theme/Doc est principalement une méthodologie permettant d'identifier des préoccupations au niveau de l'étude des exigences d'un système.

[4] respectivement merge, override, et bind en anglais

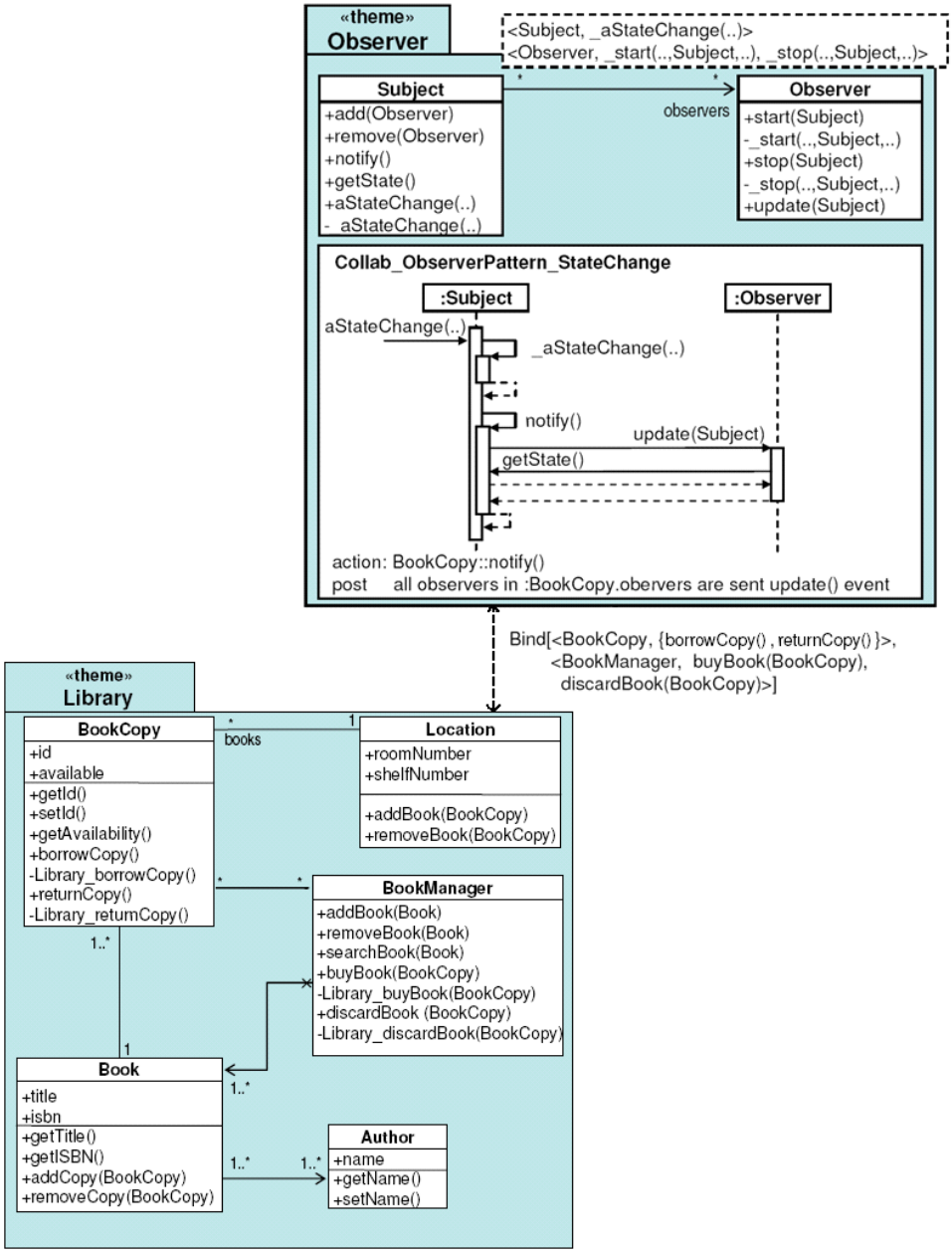


FIG. 1.4 – Le modèle d'aspect d'un patron sujet-observateur en utilisant Theme

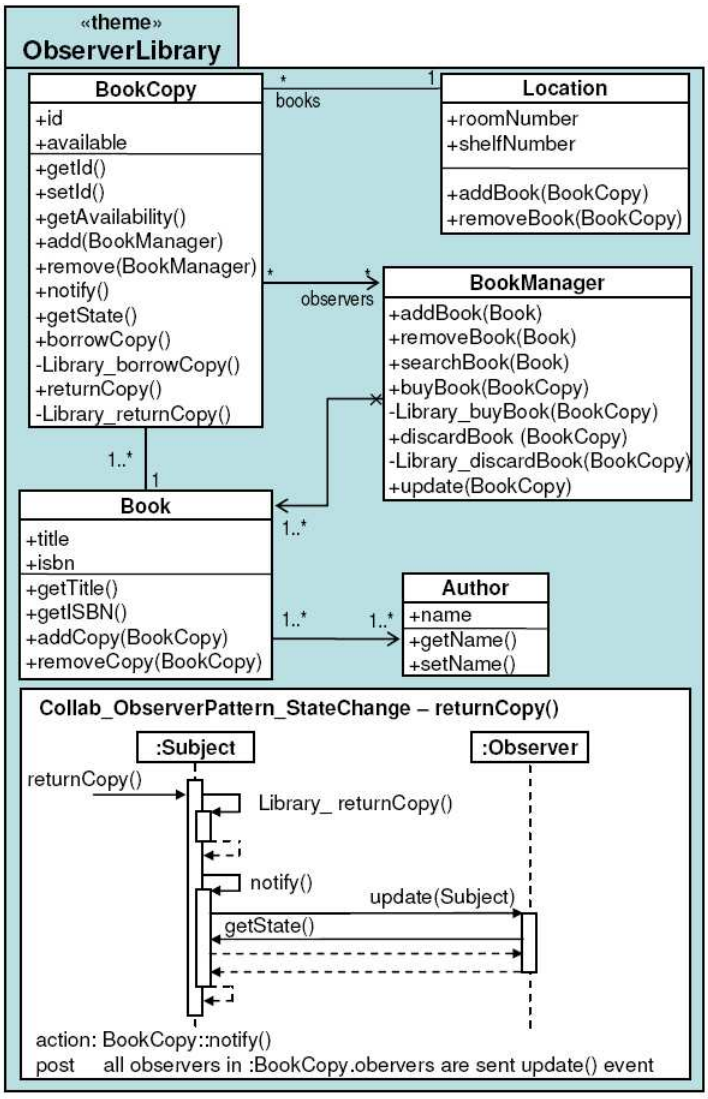


FIG. 1.5 – Le modèle tissé en utilisant la notation de Theme

1.2.3 Approche de modélisation orientée-aspect de Muller et al.

Aperçu et composition : Muller et al.[Mul06, CCMV05] proposent un approche permettant de composer des modèles génériques réutilisables pour la conception de différents systèmes d'information. Ils proposent des mécanismes de construction de systèmes par application de modèles paramétrés, où ces modèles sont eux-mêmes paramétrés par un modèle afin de permettre l'expression d'une partie requise complexe. Il est ainsi possible de définir des *composants de modèle* dont les interfaces requises et fournies sont exprimées par un modèle.

La fonctionnalité exprimée par un composant peut alors être ajoutée à un modèle par un mécanisme d'application. Ce mécanisme consiste à mettre en relation le modèle requis par le composant avec un modèle conforme auquel la fonctionnalité doit être ajoutée. Ce mécanisme fonctionne aussi bien pour l'application à un modèle de base qu'à un autre composant de modèle, permettant ainsi la construction de fonctionnalités complexes à partir de fonctionnalités plus simples. La conception d'un système peut alors être réalisée par l'assemblage d'un ensemble de composants de modèle.

Muller et al. ont également formalisé ce mécanisme d'application sous forme d'un opérateur [MCCV05]. Ils ont ainsi démontré des propriétés d'ordre permettant de garantir la cohérence des alternatives de composition. Elles permettent notamment de définir des chaînes d'applications et la possibilité d'ajouter une fonctionnalité à un système sans remise en cause des applications précédentes.

Ils ont proposé une définition stricte de leurs artefacts de modélisation, les rendant en particulier manipulables. Ils ont utilisé pour cela une technique de métamodélisation et un ensemble de contraintes exprimées à l'aide du langage OCL. Afin de permettre l'expression de leurs composants de modèle et de leur assemblage à l'aide du langage standard UML, ils ont réalisé des métamodèles par extension du métamodèle UML. Leurs composants de modèle sont ainsi définis à partir du concept de paquetage template (similaire à celui de France et al.) auquel ils ont ajouté un ensemble de contraintes afin de garantir que son paramétrage forme bien un modèle. Leur relation pour exprimer l'application d'un composant de modèle (apply) est également définie par extension du métamodèle standard UML. Les contraintes qu'ils ont définies (en OCL) permettent de garantir la conformité entre modèle requis et modèle fourni.

Conclusion : L'approche de Muller et al. n'est pas directement liée à une approche de modélisation orientée-aspect, mais comme l'approche France et al., elle propose des mécanismes de composition (qui sont de réelles fusions de modèles) essentiellement sur des modèles structuraux. L'avantage majeur de cette approche est la définition formelle d'opérateurs de composition, et la proposition de propriétés permettant de garantir la cohérence des compositions. N'étant pas directement liée à une approche de modélisation orientée-aspect, l'approche de Muller et al. ne proposent pas de mécanisme de détection de points de jonction.

Notons que dans la même équipe, Barais et al [Bar05] ont travaillé sur l'introduction de préoccupations dans la description d'une architecture logicielle à base de composants afin de faciliter la prise en compte de ces langages dans un processus de développement logiciel incrémental. Dans ces travaux, Barais et al ont mis en avant l'intérêt de posséder

à la fois des points de jonction identifiés de manière structurelle (avant, après ou autour de l'exécution d'un service) mais aussi des points de jonction identifiés de manière comportementale sous forme de séquence de messages. Cependant, la détection de séquences de messages définis dans une expression de coupe reste pour le moment naïve. En outre, la définition de la composition est définie de manière programmatique. Le langage de transformation dédié offert par l'approche TranSAT est d'ailleurs assez pauvre pour l'expression de la composition de diagrammes comportementaux [BLMD06]. Pour ces raisons, nous n'avons pas détaillé ces travaux.

1.2.4 Approche de modélisation orientée-aspect de Stein et al.

Les travaux de Stein et al. portant sur la modélisation orientée-aspect peuvent être divisés en deux catégories. La première regroupe les travaux présentés dans [SHU02c, SHU02a, SHU02b], qui ont pour objet la modélisation de programmes écrits avec le langage de programmation orientée-aspect Aspect-J. La deuxième porte sur la représentation d'expressions de coupe avec des diagrammes d'UML2.0.

Premier aperçu : Nous allons commencer par présenter les travaux permettant sur la modélisation de programme écrit avec Aspect-J. Stein et al. ont étudié Aspect-J et UML pour : premièrement, identifier les concepts communs entre UML et Aspect-J ; deuxièmement, étendre UML pour supporter les concepts d'Aspect-J absents dans UML ; finalement, pour identifier les concepts d'UML qui sont plus expressifs que ceux d'Aspect-J, comme par exemple la destruction d'instance. L'approche de Stein et al. est conçue comme une extension d'UML 1.x (dont le métamodèle est légèrement modifié). La structure d'un système ainsi que son comportement sont modélisés avec des diagrammes de classe, de collaboration et de séquence. De plus, les diagrammes de séquence sont utilisés pour représenter des points de jonction.

Dans l'approche de Stein et al., les aspects sont représentés par le stéréotype ≪ *aspect* ≫ (conf. figure 1.6) qui est dérivé de la métaclasse *Class* d'UML. En plus, plusieurs attributs permettent de capturer les particularités des aspects écrits en Aspect-J, comme *pointcut*, *advice*, etc. Les éléments à composer sont représentés avec des diagrammes de cas d'utilisation et de collaboration. L'ordre de tissage des aspects peut être spécifié avec des relations de dépendance entre aspects (avec des stéréotypes du genre ≪ *dominates* ≫).

Première conclusion : Comme l'objectif de l'approche de Stein et al. est de modéliser des programmes Aspect-J, elle ne propose pas de mécanismes de composition et de détection de points de jonction.

Deuxième aperçu : Stein et al. [SHU] ont également proposé une approche pour représenter des expressions de coupe à un niveau de modélisation, appelée JPDD (pour join point designation diagrams en anglais). L'idée intéressante de cette approche est d'offrir la possibilité d'utiliser le modèle le plus adapté à l'expression de coupe souhaitée. Par exemple, pour exprimer la volonté de détecter des séquences de messages, les diagrammes de séquences d'UML sont bien adaptés. La figure 1.7 représente deux expressions de coupe, une utilisant les diagrammes de séquences, et l'autre les diagrammes d'état. Des caractères génériques et d'autres symboles sont utilisés pour augmenter l'-

expressivité de leurs expressions de coupe.

Deuxième conclusion : Malheureusement, leur approche ne sert qu'à exprimer des expressions de coupe, mais pas à détecter les points de jonction correspondants. Un travail important portant sur la faisabilité de leurs JPDD et la proposition de mécanisme de détection correspondant, est nécessaire pour que leur approche puisse être utilisée.

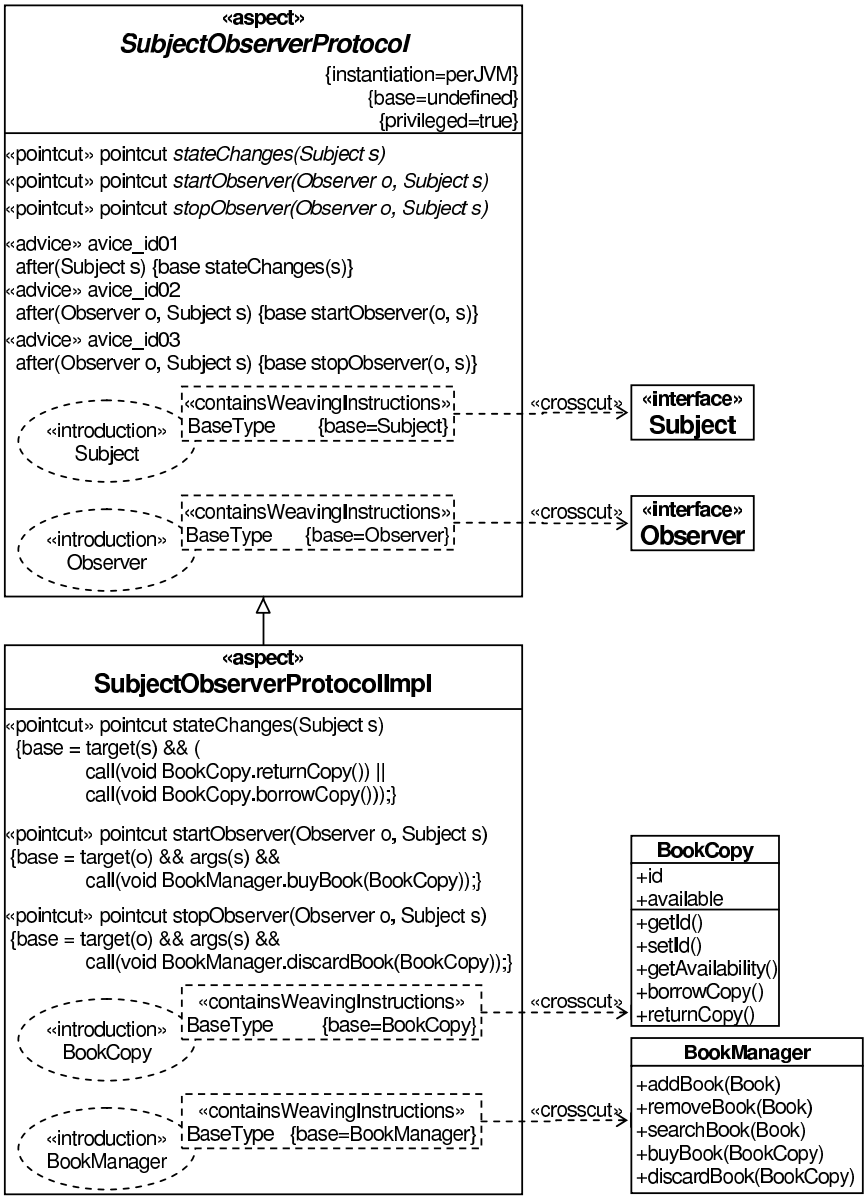


FIG. 1.6 – Le modèle d'aspect d'un patron sujet-observateur en utilisant l'approche de Stein et al. pour Aspect-J

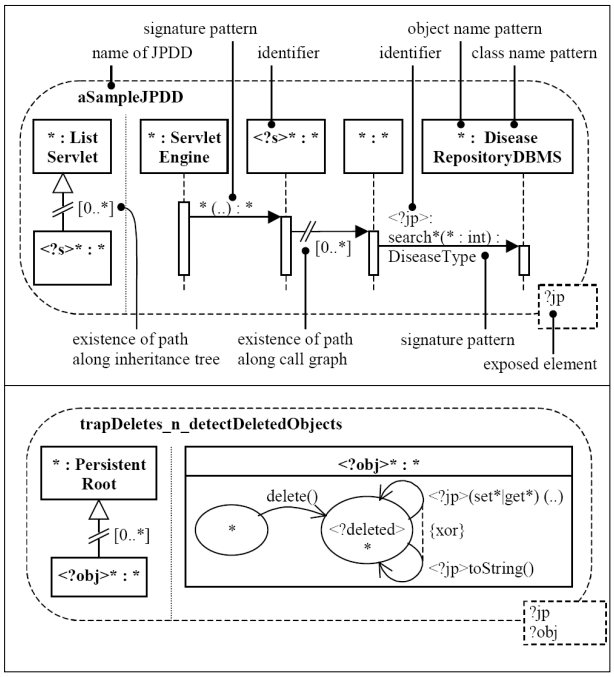


FIG. 1.7 – Deux exemples d'expressions de coupe en utilisant JPDD

1.2.5 Approche de modélisation orientée-aspect de Aldawud et al.

Aperçu : Dans [EAB05, AEB03], Aldawud et al. ont proposé un profil d'UML pour AOSD, basé sur les versions 1.x d'UML. L'état actuel de leur approche ainsi que l'influence de leurs travaux plus anciens [AEB01], nous invitent à penser que l'approche proposée est davantage une approche qui permet de modéliser un programme écrit avec un langage orienté-aspect. Autrement dit, un système peut être modélisé par aspects avec l'approche de Aldawud et al., mais le tissage des aspects ne se fera qu'au niveau d'une phase de programmation. Tandis que les diagrammes de classe sont utilisés pour exprimer la structure d'un système, les diagrammes d'état sont utilisés pour décrire le comportement de celui-ci.

Composition/Représentation : Les aspects sont représentés par le stéréotype ≪ *aspect* ≫, qui est dérivé de la métaclasse *Class* d'UML (conf. figure 1.8). L'approche permet également d'annoter les aspects en distinguant les aspects dits *synchrones*, dès aspects dits *asynchrones* qui n'ont pas d'impact sur le modèle de base (comme les aspects de trace). Si un aspect est synchrone, le profil requiert que l'aspect définisse les opérations *Preactivation* et *Postactivation*, dont le but est de synchroniser le comportement du modèle d'aspect avec le comportement du modèle de base. L'annotation *synchrone* permet également de spécifier des conflits possibles et des mécanismes pour les résoudre. La spécification du comportement de l'aspect est établie par l'utilisation de diagrammes d'état (figure 1.9), où les comportements "transverses" sont modélisés par des événements qui déclenchent une transition.

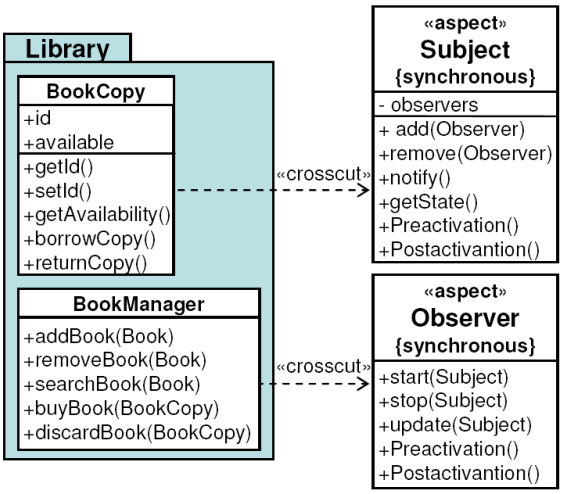


FIG. 1.8 – Le modèle d'aspect d'un patron sujet-observateur en utilisant le profile d'Aldawud et al.

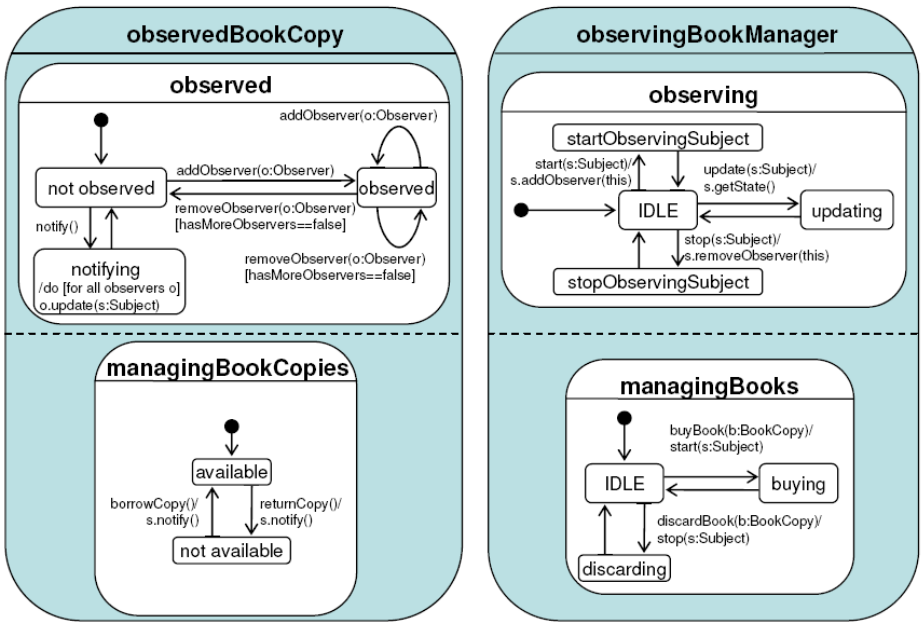


FIG. 1.9 – Le comportement transverse de l'observateur avec la représentation d'Aldawud et al.

Conclusion : L'approche de Aldawud et al. est principalement utilisée pour représenter des aspects à un niveau de modélisation. Mais une fois encore, cette approche ne propose pas de réels mécanismes de tissages d'aspects à un niveau de modélisation.

1.2.6 Synthèse

Nous aurions pu présenter plusieurs autres approches, telles que celle de Pawlak et al. [PDF+02, PSD+05][5], Ortiz et al. [OHCA05][6], etc. Mais aucune des ces approches et aucune de celles présentées, à l'exception des approches de France et al. et de Muller et al., ne présente de mécanismes de tissage définis formellement (ou implantés) permettant de tisser des aspects à un niveau de modélisation.

Les approches de France et Muller sont les seules à présenter des mécanismes de composition bien définis, mais uniquement pour des diagrammes structuraux comme les diagrammes de classe A notre connaissance, il n'existe donc pas de tisseur d'aspects pour des modèles dynamiques tels que les diagrammes de séquence, les Message Sequence Chart, ou encore les diagrammes d'état.

Nous pouvons avancer au moins une explication à ce manque. Premièrement, le domaine de la modélisation orientée-aspect est relativement jeune (environ 5 ans). Or on peut supposer, que le premier problème que les chercheurs ont essayé de résoudre avant de tisser des modèles d'aspects était simplement de représenter des aspects à un niveau de modélisation. Dans ce sens, plusieurs approches avaient pour but unique de pouvoir modéliser des programmes écrits avec des langages orientés-aspect (notamment avec Aspect-J) et le tissage des aspects avait lieu uniquement à un niveau de programmation.

Il est vrai qu'une possibilité est de simplement représenter les aspects à un niveau de modélisation, et de garder la séparation des aspects jusqu'à la phase d'implantation où les aspects sont finalement tissés avec des langages de programmation orientés-aspect. Nous identifions au moins trois critiques majeurs à cette approche. Premièrement, certains aspects peuvent être relativement facilement tissés à un niveau de modélisation lorsqu'ils sont spécifiés avec un langage de modélisation donné, alors qu'il est très difficile de les tisser, voire même impossible, à un niveau de programmation. Par exemple, il est très difficile de tisser des aspects à un niveau de programmation lorsque l'expression de coupe représente des séquences de messages. Deuxièmement, il est parfois difficile de garder exactement la même décomposition d'un système à un niveau de modélisation et à un niveau de programmation, car les concepts utilisés sont différents. Troisièment, si on tisse les aspects uniquement à un niveau de programmation, cela signifie que le système développé ne peut être vu en globalité qu'à la fin du développement. Il est donc possible qu'une erreur produite durant une phase de modélisation ne puisse être détectée que bien plus tard. Ce qui pose des problèmes de localisation d'erreurs et rend la correction bien plus difficile et coûteuse.

Pour les raisons qui viennent d'être invoquées, mais aussi pour les motivations présentées en introduction de cette thèse et dans la section précédente, notre objectif est de fournir un tisseur d'aspects pour des modèles dynamiques, en particulier pour des modèles de scénarios (Message Sequence Chart ou diagramme de séquence UML2.0). Ces travaux pourraient avantageusement compléter les travaux de France et al, qui proposent des compositions de diagrammes structurels, ou de Clarke et al., qui

[5] Approche dont le but principal est de modéliser des programmes écrits avec le langage orienté-aspect JAC [JAC, PSD+04]

[6] Approche visant à modéliser des aspects dans le domaine des services web

proposent une méthodologie intéressante de développement par aspects, pour fournir un environnement complet de modélisation orientée-aspect.

1.3 Formalisme d'un langage de scénarios : les Message Sequence Charts

Nous allons nous intéresser à un type de modèles comportementaux particuliers : les Message Sequence Charts (MSCs). Cette section a pour but d'introduire le formalisme des MSCs que nous utiliserons, ainsi que de présenter certaines définitions qui nous seront utiles dans le reste de la thèse. Finalement, nous présenterons, entre autres, les travaux existants portant sur la correspondance et la composition de MSCs.

1.3.1 Les MSCs

Un des formalismes les plus utilisés pour décrire informellement le comportement d'entités et les communications dans un système distribué est le chronogramme. L'écoulement du temps sur chaque processus est symbolisé par un trait vertical, les messages par des flèches de l'émetteur vers le récepteur. De nombreuses documentations de protocoles de communication contiennent ces représentations, qui servent à illustrer par un exemple le comportement d'un système.

Les Message Sequence Charts (ou MSCs) sont un formalisme qui permet de spécifier graphiquement le comportement d'entités communicantes. Initialement, les MSCs n'étaient utilisés qu'en tant que traces de sortie pour des systèmes distribués spécifiés en SDL. L'idée de normaliser ce langage graphique s'est rapidement imposée, et un groupe d'étude de l'ITU a été chargé de la question. Plusieurs documents ont ainsi vu le jour, proposant les langages successifs MSC'92 [ITU93], MSC'96 [ITU96] et MSC'2000 [ITU99]. D'après ces documents, les MSCs peuvent servir à :

- donner une vision d'un service fourni par plusieurs entités,
- exprimer des exigences,
- comme base pour l'élaboration, la simulation et la vérification de systèmes SDL,
- pour exprimer des jeux de test,
- pour spécifier des communications ou des interfaces.
- comme une formalisation des cas d'utilisation dans les méthodes orientées-objet.

Dans cette thèse nous allons utiliser les MSC'96, les MSC'2000 introduisant principalement des notions de temps (timer, ...) que nous ne considérerons pas. Les MSC'96 permettent de définir un système hiérarchiquement. Au plus bas niveau, des Basic Message Sequence Charts (bMSCs) décrivent le comportement de processus finis. Les bMSCs sont ensuite composés par plusieurs niveaux hiérarchiques de graphes : les High Level Message Sequence Charts (HMSCs).

1.3.2 Les bMSCs

Dans un bMSC, les entités d'un système, appelées instances, sont représentées par des axes verticaux. Les échanges de messages sont représentés par des flèches étiquetées

par des noms de messages, de l'instance émettrice, à l'instance réceptrice. Les messages sont supposés asynchrones. Un bMSC définit un ensemble d'événements qui sont les occurrences d'actions dans le système (émission et réception de messages), et une relation de précédence sur ces événements : une émission de message doit toujours précéder la réception correspondante, et les événements sont totalement ordonnés le long d'une instance (excepté dans les parties que l'on appelle corégion). La figure 1.10 représente un bMSC nommé *log* qui représente un comportement où les messages *log in* et *accept* sont échangés entre les instances *client* et *serveur*. Chaque message est composé de deux événements : un envoi et une réception de message. Par exemple, le message *log in* est composé des événements e_1 et e_3.

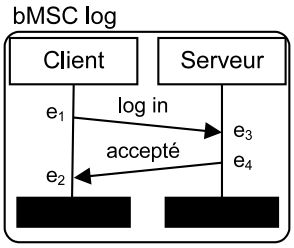


FIG. 1.10 – Exemple d'un bMSC

Une sémantique d'algèbre de processus à été proposée pour les bMSCs dans [Ren98], mais il semble plus naturel de donner aux bMSCs une sémantique nom entrelacée comme dans [HJC98] et [KL98]. Par la suite, notre définition formelle d'un bMSC sera fondée sur les notions de préordre et d'ordre partiel. Un *préordre* sur un ensemble d'éléments E est une relation $R \subseteq E^2$ qui est reflexive (c'est-à-dire $\forall e \in E, eRe$) et transitive ($\forall e, f, g \in E, eRf \wedge fRg \Longrightarrow eRg$). Un *ordre partiel* est un préordre antisymétrique ($\forall a, b \in E, aRb \wedge bRa \Longrightarrow a = b$).

Comme déjà mentionné, un bMSC définit un ordre causal entre les envois et les réceptions de messages, et un ordre partiel sur les événements situés sur la même instance (en tenant compte des corégions). Les bMSCs peuvent être formellement définis par :

Définition 1.1 (Basic MSC) *Un* bMSC *est un nuplet* $M = (I, E, \leq, A, \alpha, \phi, \prec)$, *où* I *est un ensemble d'instances,* E *est un ensemble d'événements (envois et réceptions de message),* $\leq$ *est un préordre sur* E *imposé par les instances et les messages,* A *est un ensemble d'actions,* $\alpha : E \rightarrow A$ *relie les événements aux actions,* $\phi : E \rightarrow I$ *relie les événements aux instances, et* $\prec \subseteq E \times E$ *est une fonction associant une émission de message et la reception correspondante, telle que* $\prec \subseteq \leq$.

Par la suite, nous appellerons parfois *nom de message*, une action. Notons que, sans perte de généralité, nous n'utiliserons pas d'*action locale*, une *action locale* étant remplacée par un envoi et une réception de message localisés sur la même instance.

On utilisera également une fonction de typage $T : E \rightarrow \{envoi, réception\}$ qui retourne le type d'un événement.

Nous noterons par $min(E) = \{e \in E | \forall e' \in E, e' \leq e \Rightarrow e' = e\}$, l'ensemble des événements minimaux, c'est-à-dire l'ensemble des événements n'ayant d'autres prédécesseurs qu'eux-mêmes.

Nous noterons par $\widehat{\downarrow}_{\leq,E}(e) = \{e' \in E | e' \leq e\}$, l'ensemble des prédécesseurs de l'événement e, et par $\widehat{\uparrow}_{\leq,E}(e) = \{e' \in E | e \leq e'\}$, l'ensemble des successeurs de e. Ces deux notations peuvent être utilisées pour un sous-ensemble E' d'un ensemble E : $\widehat{\downarrow}_{\leq,E}(E') = \{e \in E | \exists e' \in E', e \leq e'\}$ et $\widehat{\uparrow}_{\leq,E}(E') = \{e \in E | \exists e' \in E', e' \leq e\}$. Nous utilisons également des flèches sans chapeau définies par : $\downarrow_{\leq,E}(e) = \widehat{\downarrow}_{\leq,E}(e) - \{e\}$ qui représente l'ensemble des prédécesseurs de e excepté e[7], et $\uparrow_{\leq,E}(e) = \widehat{\uparrow}_{\leq,E}(e) - \{e\}$ qui représente l'ensemble des successeurs de e excepté e. En abusant légèrement de la notation, quand M' est un bMSC inclus dans un bMSC M, on notera par $\widehat{\uparrow}(M')$ l'ensemble des événements de M' plus les événements précédents ceux de M'.

Voici maintenant quelques définitions portant sur les bMSCs.

Définition 1.2 (BMSC bien formé) *Nous dirons qu'un bMSC M est bien formé si $\leq$ est une relation d'ordre (c'est-à-dire une relation antisymétrique).*

Définition 1.3 (Corégion) *Une corégion est une zone spécifique d'une instance, représentée par des pointillés, où les événements ne sont pas ordonnés.*

La figure 1.11 montre un exemple de corégion. Les événénements e_{14} et e_{23} situés sur cette corégion ne sont pas ordonnés, mais ils précédent tous les deux l'événement e_{04} et succèdent à l'événement e_{03}.

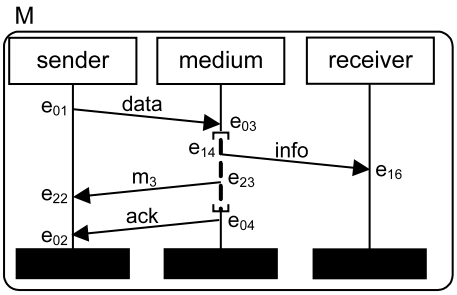


FIG. 1.11 – Exemple d'une corégion

Définition 1.4 (Croisement) *Soient deux événements e et f d'un bMSC $M = (I, E, \leq, A, \alpha, \phi, \prec)$ tel que $e \prec f$. Nous dirons que le message formé par e et f croise un ensemble d'événements E' de M si et seulement si*

- *$e \notin E' \wedge f \notin E'$;*
- *$e \in \downarrow_{\leq,E}(E') \wedge f \in \uparrow_{\leq,E}(E')$ ou bien $e \in \uparrow_{\leq,E}(E') \wedge f \in \downarrow_{\leq,E}(E')$ (e appartient à l'ensemble des prédécesseurs de E' et f à l'ensemble des successeurs ou inversement).*

[7]Nous rapellons que e est son propre prédécesseur car $e \leq e$

La figure 1.12 montre un exemple où le message m_0 croise l'ensemble d'événements formé par les messages m_1 et m_2.

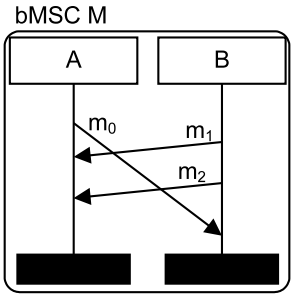


FIG. 1.12 – Un exemple de croisement

Définition 1.5 (BMSCs indépendants) *Deux bMSCs $M1$ et $M2$ sont indépendants, noté $M1||M2$, si et seulement si $M1 \bullet M2 = M2 \bullet M1$ (cette situation a lieu uniquement quand $\phi(E_1) \cap \phi(E_2) = \emptyset$).*

Définition 1.6 (BMSC connexe) *Un bMSC M est connexe si et seulement si il n'existe pas deux bMSCs $M1$ et $M2$ (différent de M_ϵ) tel que $M = M1 \bullet M2$ et $M1||M2$.*

Définition 1.7 (BMSC clos pour les communications) *Un bMSC $M = (I, E, \leq, A, \alpha, \phi, \prec)$ est clos pour les communications si et seulelement si $\forall e \in E, e \prec f \Rightarrow f \in E$ et $\forall f \in E, e \prec f \Rightarrow e \in E$.*

Définition 1.8 (BMSC atomique) *Un bMSC B est appelé un atome s'il n'existe pas deux bMSCs $N \neq M_\epsilon$ et $M \neq M_\epsilon$ tel que $B = M \bullet N$.*

Pour un ensemble de bMSCs $\mathcal{M}$, nous notons par $At(\mathcal{M})$ l'ensemble de tous les atomes des bMSCs de $\mathcal{M}$. Notons qu'un algorithme de décomposition d'un bMSC en atomes peut être trouvé dans [HLM00].

Définition 1.9 (Factorisation d'un bMSC) *Pour un bMSC M tel que $M = X_1 \bullet X_2 \bullet \cdots \bullet X_k$, où tous les X_i sont des bMSCs, $X_1 \bullet X_2 \bullet \cdots \bullet X_k$ est appelée une factorisation de M.*

Dans [HLM00], Helouet et al. ont proposé une notion de coupe de bMSCs, utile pour décomposer un bMSC.

Définition 1.10 (Point de coupe d'un bMSC) *Dans un bMSC $M = (I, E, \leq, A, \alpha, \phi, \prec)$, un point de coupe est un sous-ensemble $cp \subseteq E$ tel que :*

- *$\exists i \in I, \forall e \in cp, \phi(e) = i$ (tous les éléments de cp sont situés sur la même instance)*
- *$\forall e \in cp, \forall e' \leq e$, si $\phi(e') = \phi(e)$ alors $e' \in cp$, (tous les prédécesseurs d'un événement de cp sur la même instance appartiennent également à cp)*

– $\forall e \in cp, \{e' \in E, \phi(e') = \phi(e) \wedge e \not\leq e' \wedge e' \not\leq e\} \subseteq cp$

Un point de coupe définit un *point*, sur la représentation graphique des instances, situé entre deux événements. Notons que cette définition ne permet pas qu'un point de coupe soit placé dans une corégion.

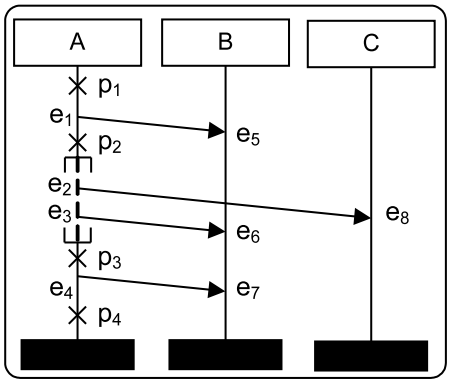


FIG. 1.13 – Illustration des points de coupe

Sur l'exemple de la figure 1.13, les points de coupe sur l'instance A sont : $I_1 = \emptyset, I_2 = \{e_1\}, I_3 = \{e_1, e_2, e_3\}, I_3 = \{e_1, e_2, e_3, e_4\}$ qui définissent les points p_1, p_2, p_3, p_4 sur l'axe de l'instance (représenté par les croix).

Définition 1.11 (Coupe d'un bMSC) *Une coupe d'un bMSC $M = (I, E, \leq, A, \alpha, \phi, \prec)$ est un sous-ensemble C de E tel que $C = \cup_{i \in I} cp_i$ est l'union des points de coupe associés à chaque instance.*

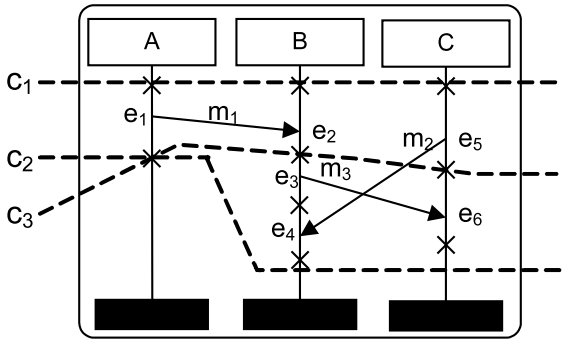


FIG. 1.14 – Illustration des quelques coupes

Une coupe peut être représentée graphiquement par une ligne traversant les points définis par les points de coupe. La figure 1.14 montre trois coupes possibles C_1, C_2 et C_3. Mais toutes ces coupes ne permettent pas de découper un bMSC en deux bMSCs clos pour les communications, autrement dit, ces coupes peuvent provoquer des coupures de messages..

Définition 1.12 (Coupe valide d'un bMSC) *Une coupe partitionne un bMSC M en deux ensembles $M_1 = (C, \leq_{|C})$ et $M_2 = (E \backslash C, \leq_{E \backslash C})$. Une coupe C sera qualifiée de valide si M_1 et M_2 sont deux bMSCs clos pour les communications.*

Autrement dit, une coupe valide d'un bMSC M est une coupe qui partitionne M en deux autres bMSCs sans couper de messages. Dans l'exemple de la figure 1.14, C_3 n'est pas une coupe valide car le message m_2 est coupé.

1.3.3 Les High-Level Message Sequence Charts : HMSCs

Les bMSCs seuls n'ont pas un pouvoir expressif suffisant : ils ne définissent que des comportements finis sans réelles alternatives. Pour cette raison, les MSCs ont été étendus par des MSCs de Haut niveau (HMSC), c'est-à-dire par un plus haut niveau de spécification qui permet la composition de bMSCs avec des opérateurs tels que la séquence, l'alternative et la boucle.

La figure 1.15 montre deux représentations d'un HMSC *log*. La représentation de gauche est plus abstraite, elle décrit la composition de bMSCs en faisant référence à ces bMSCs, alors que la représentation de droite contient la description détaillée de chaque bMSC. Sur cette figure, nous pouvons remarquer que par exemple, les bMSCs *log* et *essai* sont composés séquentiellement, que les bMSCs *ok* et *non* sont composés alternativement, etc. Les HMSCs peuvent être considérés comme des automates étiquetés par des bMSCs. Ils peuvent être définis formellement par :

Définition 1.13 (HMSC) *Un* HMSC *est un graphe $H = (\mathcal{N}, T, q_0, \mathcal{Q}_{end}, \mathcal{M})$ où : $\mathcal{N}$ est un ensemble de noeuds, $T \subseteq \mathcal{N} \times \mathcal{M} \times \mathcal{N}$ est un ensemble de transitions, q_0 est un noeud initial, $\mathcal{Q}_{end}$ est un ensemble de noeuds finaux, et $\mathcal{M}$ est un ensemble fini de bMSCs.*

Pour une transition $t = (p, l, q) \in T$, le noeud p est appelé l'origine de la transition t et noté $\lambda_-(t)$. Le noeud q est appelé la destination de t et noté $\lambda_+(t)$. l est appelé l'étiquette de t et notée $\lambda(t)$. De plus, nous supposerons qu'aucune transition ne peut quitter un noeud final.

Parfois, pour simplifier la notation d'un HMSC, nous pouvons simplement représenter un HMSC par un automate étiqueté par des noms de bMSCs. Par exemple, le HMSC de la figure 1.15 peut être représenté par l'automate de la figure 1.16.

La notion de composition séquentielle[8] (notée $\bullet$) est centrale pour comprendre la sémantique des HMSCs. Elle correspond à la concaténation locale définie dans [Pra86] sur les ordres[9], et plus tard dans [RW94] sur des algèbres de processus. La figure 1.17 donne un exemple de cet opérateur. De manière imagée, la composition séquentielle de deux bMSCs consiste à coller les deux diagrammes suivant les axes de leurs instances communes. Notons que l'opérateur de séquence impose uniquement un ordre sur les

[8]Nous utilisons la composition séquentielle faible. Nous rappelons qu'il existe une composition séquentielle forte pour laquelle dans la séquence de deux bMSCs M_1 et M_2, tous les événements de M_1 doivent être terminés avant qu'un événement de M_2 ne soit exécuté.

[9]Plus précisèment, c'est une spécialisation de l'opérateur de Pratt.

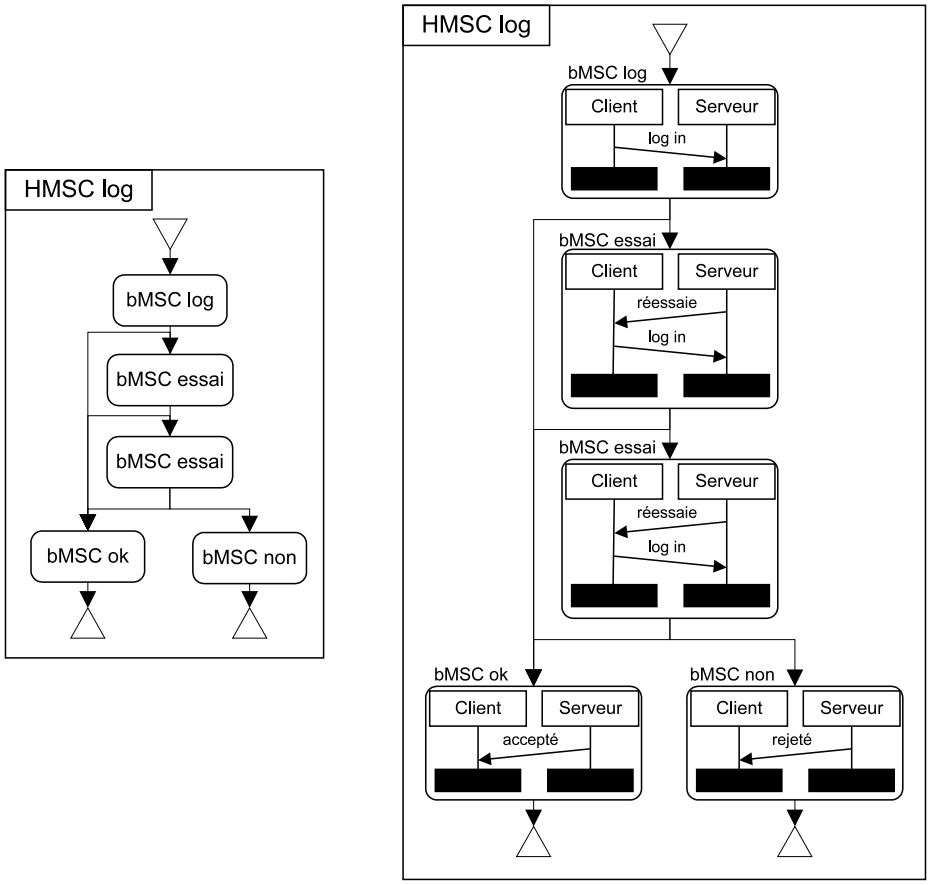


FIG. 1.15 – Exemple d'un HMSC

événements situés sur une même instance, mais que les événements situés sur des instances différentes dans deux bMSCs $M1$ et $M2$ peuvent être concurrents dans $M1 \bullet M2$. La composition séquentielle peut être définie formellement par :

Définition 1.14 (Composition Séquentielle) *La* composition séquentielle *de deux bMSCs* M_1 *et* M_2 *est le bMSC* $M_1 \bullet M_2 = (E_1 \uplus E_2, \leq_{1\bullet 2}, \alpha_1 \cup \alpha_2, \phi_1 \cup \phi_2, A_1 \cup A_2, \prec_1 \uplus \prec_2)$, *où :* $\leq_{1\bullet 2} = \left(\leq_1 \uplus \leq_2 \uplus \{(e_1, e_2) \in E_1 \times E_2 \mid \phi_1(e_1) = \phi_2(e_2)\}\right)^*$

Pour calculer le nouvel ordre partiel $\leq_{1\bullet 2}$, la composition séquentielle ordonne les événements e_1 d'un bMSC M_1 et e_2 d'un bMSC M_2 s'ils sont situés sur la même instance, et calcule ensuite la fermeture transitive de l'ordre. Dans cette définition, $\uplus$ est l'union disjointe de deux ensembles, c'est-à-dire une opération d'union où les éléments communs des deux ensembles sont dupliqués.

Par la suite, nous noterons également M^k la concaténation séquentielle de k copies d'un bMSC M, et par M^* l'ensemble $\bigcup_{i\in\mathbb{N}} M^i$ avec la convention que $M^0 = M_\epsilon$ (M_ϵ étant le bMSC vide).

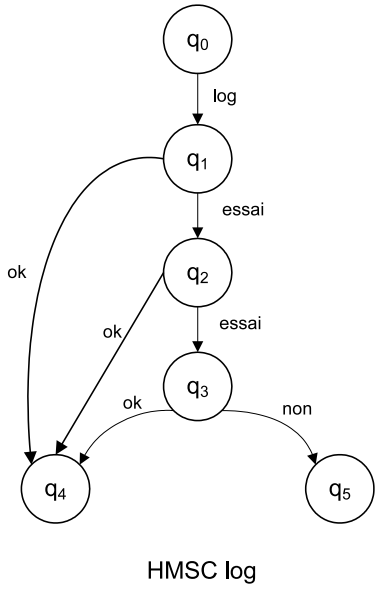


FIG. 1.16 – Exemple de représentation en automate du HMSC de la figure 1.15

Un *chemin initial* d'un HMSC H est une séquence de transitions $p = t_1.t_2...t_k$ tel que l'origine de t_1 est le noeud initial q_0. Si de plus, la destination de t_k est un noeud présent dans $\mathcal{Q}_{end}$, alors p est appelé un *chemin accepteur*. Le bMSC $\lambda(p)$ associé à un chemin p est la concaténation séquentielle des bMSCs étiquetant les transitions, $\lambda(p) = \lambda(t_1) \bullet \lambda(t_2) \bullet ... \bullet \lambda(t_k)$.

Nous pouvons maintenant définir le langage ou la sémantique d'un HMSC par :

Définition 1.15 (Langage ou sémantique d'un HMSC) *Le langage ou la sémantique d'un HMSC H est l'ensemble des comportements définis par les bMSCs étiquetant tous les chemins accepteurs, c'est-à-dire l'ensemble de bMSCs $\mathcal{L}(H) = \{\lambda(p) | p$ étant un chemin accepteur de $H\}$.*

Un HMSC peut alors être vu comme un générateur d'un ensemble potentiellement infini de bMSCs. Par exemple, le HMSC *log* de la figure 1.15 génère l'ensemble de bMSCs : $\mathcal{L}(log) = \{log \bullet ok, log \bullet essai \bullet ok, log \bullet essai \bullet essai \bullet ok, log \bullet essai \bullet essai \bullet non\}$. Bien entendu, l'ensemble de bMSCs généré par un HMSC est infini dès que le HMSC contient un cycle.

Notons que dans [MPS98], Muscholl et al. montre les relations entre HMSCs et les traces de Mazurkiewicz. Il en résulte que pour deux HMSCs H_1, H_2, les propriétés suivantes sont indécidables :

- $\mathcal{L}(H_1) = \mathcal{L}(H_2)$
- $\mathcal{L}(H_1) \subseteq \mathcal{L}(H_2)$
- $\mathcal{L}(H_1) \cap \mathcal{L}(H_2) = \emptyset$

Notons finalement que plus de détails sur les MSCs et des résultats s'y rapportant peuvent être trouvés dans la thèse de Genest [Gen04].

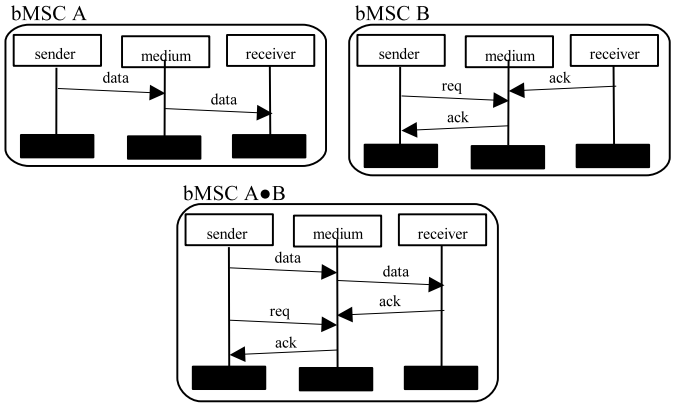


FIG. 1.17 – Composition séquentielle de A et B

1.3.4 Quelques problèmes classiques.

Dans cette partie, nous allons présenter quelques problèmes relatifs aux MSCs. Nous commencerons par présenter le problème des choix non locaux dans les HMSCs. Dans le chapitre 4, nous proposons un moyen de résoudre ce problème. Ensuite, nous présenterons les travaux existants portant sur la correspondance et la composition de MSCs, qui sont des concepts proches de ceux proposés dans cette thèse. Enfin, nous présenterons une brève comparaison entre les MSCs et les diagrammes de Séquence d'UML2.0.

1.3.4.1 Choix non local

Les choix dans les HMSC[10] sont des éléments importants : ils permettent de commencer à définir la structure de contrôle des protocoles en cours de spécification. Cependant, ces choix peuvent se révéler ambigus, notamment lorsque le contrôle n'est pas localisé sur un seul processus.

Définition 1.16 *Un choix dans un HMSC H est un noeud c comportant plus d'un successeur. Tout chemin issu d'un noeud de choix c sera appelé* alternative *de c. Tout chemin sans cycle issu d'un choix c sera appelé une* branche *de c.*

Un choix est non local lorsqu'il existe au moins deux scénarios possibles à partir d'un choix, et que plus d'une instance est responsable du comportement choisi. Sur l'exemple de la figure 1.18, lorsque l'une des instances A ou B choisit un scénario, l'autre instance doit se conformer à ce choix. La sémantique des MSCs suppose donc une synchronisation implicite entre A et B.

Lorsque l'on souhaite définir un ensemble de comportements au moyen d'un HMSC, ce genre de spécification n'est pas particulièrement choquant : seuls les comportements

[10]cette description du problème des choix non locaux est extraite de la thèse de Helouet [HÓ0], qui présente également d'autres problèmes tels que le *race problem* [AHP96], la divergence dans les HMSCs ou encore le problème de confluence [MP99].

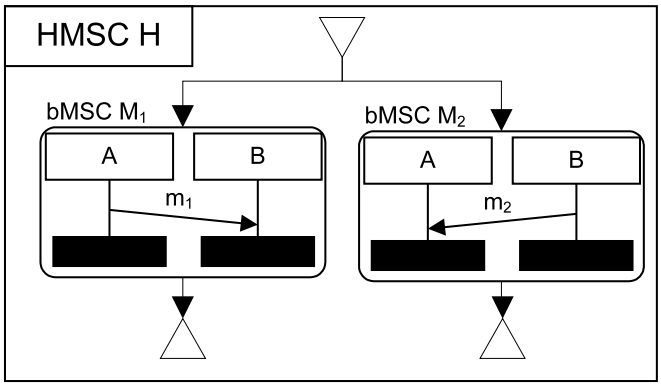


FIG. 1.18 – Un HMSC non local

décrits par chaque alternative sont possibles. Par contre, lorsqu'une implémentation est envisagée, les choix non locaux peuvent provoquer une erreur d'interprétation. En effet, plusieurs sens peuvent être associés à la description de la figure 1.18 :

- Seul les deux scénarios $M1$ et $M2$ sont possibles, il faudra donc ajouter des communications entre A et B pour éviter les croisements de message. Le modèle aura donc besoin d'être raffiné.
- Un troisième scénario, dans lequel $m1$ et $m2$ se croisent est possible. Dans ce cas, il faut définir un nouveau comportement qui contient ce croisement [AEM00].

Comme on le voit, les choix non locaux sont source d'ambiguïté, et il est important d'en donner une définition, et d'outiller les MSCs pour les détecter Une définition du choix non local a été donnée dans [BAL97]. Cette définition suppose que toute instance doit émettre ou recevoir un message dans chaque branche d'un choix, et que la séquence de bMSCs est une composition séquentielle forte. Cette supposition limite la recherche des choix non locaux aux arcs sortants d'un noeud de choix. Cependant, si l'on considère la composition séquentielle faible de bMSC comportant des ensembles d'instances disjoints, la présence de choix non locaux devient une propriété globale du HMSC.

Dans [HJ00], Hélouët et al. donnent alors une autre caractérisation du choix local qui est :

Définition 1.17 *Soient H un HMSC, et c un noeud de choix de H. c est un choix local si et seulement si :*

- $\forall p_1 = c.n_1 \dots n_k$, *chemin acyclique maximal de H* $(c \notin n_1 \dots n_k)$ *issu de c,*
- $\forall p_2 = c.n'_1 \dots n'_k$, *chemin acyclique maximal de H* $(c \notin n'_1 \dots n'_k)$ *issu de c,*

$$\forall e_1 \in min(\lambda(p_1)), \forall e_2 \in min(\lambda(p_2)), \phi(e_1) = \phi(e_2).$$

Plus intuitivement, un choix c est local si et seulement si, pour tout chemin issu de c, il existe un événement minimal unique, et que sur chaque chemin, cet événement est situé sur la même instance. Cette instance particulière sera alors appelée instance responsable du choix c. Dans [HJ00], Hélouët et al. donnent un algorithme pour détecter les choix non locaux d'un HMSC.

1.3.4.2 Correspondance de MSCs

Le problème de la correspondance d'un motif, appelé *template*, et d'un HMSC consiste à décider si un ordre incomplet est contenu dans un HMSC.

Un template $M = (I_M, E_M, \leq_M, A_M, \alpha_M, \phi_M, \prec_M)$ est en correspondance avec un bMSC $N = (I_N, E_N, \leq_N, A_N, \alpha_N, \phi_N, \prec_N)$ si et seulement si :

- $I_M \subseteq I_N$
- il existe une fonction h mettant en correspondance les événements de M et de N, telle que :
  - $\forall e \in E_M, \phi(e) = \phi(h(e))$, et
  - e et $h(e)$ sont des événements du même type (émission, réception)
  - l'ordre causal est préservé par $h : e_1 \leq e_2 \Rightarrow h(e_1) \leq h(e_2)$

En d'autres termes, un template M est en correspondance avec un bMSC N s'il est possible d'associer à tout événement de M un événement d'un même type situé sur la même instance, en préservant l'ordre causal sur l'image de M. L'inclusion de templates s'étend aux HMSCs.

Un HMSC template H est en ou-correspondance avec un HMSC G s'il existe un chemin maximal p_H de H et un chemin maximal p_G de G tels que $\lambda(p_H)$ soit en correspondance avec $\lambda(p_G)$.

Un template motif H est en et-correspondance avec un HMSC G s'il existe un chemin maximal p_G de G tel que pour tout chemin maximal p_H de H, $\lambda(p_H)$ soit en correspondance avec $\lambda(p_G)$.

Les problèmes de correspondance ont été prouvés décidables par Muscholl et al. [MPS98, MP99, Mus99]. Ils ne doivent pas être confondus avec la recherche de chemins maximaux définissant exactement le même ordre, qui est indécidable.

Les problèmes de correspondance sont relativement proches du problème de détection que nous présenterons dans les chapitres 2 et 3. Mais une caractéristique majeure différencie ces deux types de problèmes. Premièrement, dans la correspondance de templates, le problème revient à chercher s'il existe *au moins un* morphisme (injectif) du template vers une spécification. Dans le problème de détection introduit dans cette thèse, nous cherchons *tous* les morphismes d'un template vers une spécification. Nous introduisons également différentes sémantiques en modifiant les propriétés de l'image des morphismes trouvés.

1.3.4.3 Composition de MSCs

Un inconvénient des scénarios est qu'une description d'un système est généralement spécifiée par un ensemble de scénarios, qui représentent des utilisations typiques du système pour des situations données, tout en comportant des redondances. Le comportement attendu d'un système peut alors être vu comme une combinaison de toutes ces vues. Jusqu'à maintenant, il n'existe pas d'opérateur de composition satisfaisant réalisant cette combinaison. Les MSCs sont équipés d'alternatives, de séquences ou de compositions parallèles, mais ces opérateurs ne permettent pas de capturer la notion de redondance qui peut exister entre les vues composées.

Une solution est de rassembler de manière cohérente les vues à composer comme le proposent les Live Sequence Charts [HM03]. L'idée principale est de combiner les scénarios durant l'exécution. Les LSCs sont exécutés en parallèle, et les événements qui peuvent être exécutés dans plusieurs vues sont synchronisés quand c'est possible. A partir d'un ensemble initial de LSCs, un événement est exécuté, et de nouveaux scénarios sont "déclenchés" par cette exécution d'événement. Deux scénarios sont déclarés inconsistants s'ils se contredisent l'un l'autre sur l'ordre des événements communs. Le principal inconvénient de cette approche est que l'inconsistance entre scénarios ne peut être découverte que si les simulations accomplies passent à travers une configuration d'erreur.

Comme l'objectif de cette thèse est de fournir un tisseur statique (avant l'exécution) de scénarios, qui produit un scénario en sortie, ce type de composition ne convient pas.

1.3.4.4 Comparaison entre MSCs et DSs d'UML 2.0

Nous allons uniquement fournir une comparaison rapide entre les MSCs et les diagrammes de séquence (DSs) d'UML2.0. Une comparaison détaillée entre les deux langages peut être trouvée dans [Hau04]. Nous voulons principalement insister sur le fait que la sémantique des DSs d'UML2.0 est largement inspirée de celle des MSCs. Pour cette raison, si l'on se restreint aux opérateurs de composition classiques que sont la séquence, l'alternative et l'itération, on peut considérer, à la représentation graphique près, que les deux langages sont les mêmes.

En effet, les DSs d'UML2.0 ont sensiblement amélioré les versions précédentes de scénarios proposés dans UML1.x. Les DSs décrivent des interactions entre un ensemble d'objets d'un système et ces interactions sont maintenant considérées comme une collection d'événements (au lieu d'une collection de messages dans UML1.x). De plus, les DSs peuvent maintenant être composés aux moyens d'opérateurs pour obtenir des interactions plus complexes et infinies, à travers une représentation sous forme d'automate appelé Interaction Overview Diagram, similaire aux HMSCs. Notons tout de même que certains opérateurs des DSs d'UML2.0 n'ont pas de réelle sémantique.

1.4 Aspects Comportementaux

Dans [FF00], Filman et Friedman proposent une définition générale de ce que peut être un aspect. Il est maintenant admis qu'une technologie orientée-aspects peut être comprise comme le désir de faire des formulations quantifiées sur le comportement d'un programme (ou d'un modèle), une formulation étant typiquement de la forme :

Dans un programme (ou un modèle) P, à chaque fois que la condition C apparaît, accomplir l'action A.

La quantification est représentée par la partie "à chaque fois que la condition C apparaît", C étant appelé expression de coupe dans la terminologie aspect. L'apparition concrète de C dans le programme ou le modèle de base est appelée point de jonction , tandis que l'action A joue le rôle de ce que l'on appelle advice.

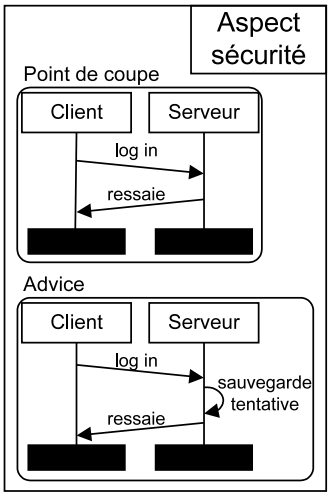


FIG. 1.19 – Exemple d'un aspect comportemental

Par analogie à cette définition, à un niveau de modélisation, des aspects comportementaux même complexes peuvent être facilement décrits à l'aide par exemple d'une paire de bMSCs. Un bMSC pour la description de l'expression de coupe, et un second bMSC pour la description de l' advice. Plus précisément, nous définissons un aspect comportemental comme une paire $A = (P, Ad)$ de bMSCs. P est une expression de coupe, c'est-à-dire un bMSC interprété comme un prédicat sur la sémantique des MSCs satisfait par tous les points de jonction. Ad est un advice, c'est-à-dire le nouveau comportement qui doit être composé avec le comportement de base au niveau des points de jonction préalablement détectés. Tout comme Aspect-J, où un aspect peut être inséré "autour", "avant" ou "après" un point de jonction, avec l'approche définie dans cette thèse, un advice pourra indifféremment compléter un point de jonction, le remplacer par un autre comportement ou le supprimer entièrement (cela sera montré au chapitre 4).

La figure 1.19 montre un exemple d'aspect comportemental : à chaque fois que l'on détecte le message *log in* suivi du message *réessaie* dans un scénario de base, le comportement décrit par l' advice doit être ajouté. (c'est-à-dire le message *sauvegarde tentative* doit être intercalé entre les messages *log in* et *réessaie*). Nous pouvons qualifier cet aspect d'aspect de sécurité, car il permet d'identifier (et ici de sauvegarder) les tentatives de "log in" qui échouent.

Nous pouvons noter que le fait d'exprimer une expression de coupe avec un bMSC, permet d'exprimer très simplement des comportements tels que des séquences de messages, chose très difficile à faire avec un langage à aspects tel que Aspect-J.

Aperçu du processus de tissage proposé dans cette thèse

Dans cette thèse, nous proposons est un tissage statique. Nous transformons un scénario en un autre scénario dans lequel les aspects ont été tissés. D'autres approches,

telles que celle de Douence et al. [DFS02, DMS01], proposent un tissage dynamique d'aspects. Le principe de l'approche de Douence et al. est de contrôler les traces d'un comportement pour tisser les aspects à la "volée". Plus particulièrement, ils interceptent des événements correspondants à des patrons de communications donnés durant l'exécution d'un système. Un des inconvénients de ce genre d'approche est de ne pouvoir agir qu'à posteriori. Il n'est pas possible de modifier un comportement se trouvant juste avant le patron d'événements détecté. Notons que les langages de programmation orientés-aspect tels qu' Aspect-J [KHH+01] ou JAC [JAC, PSD+04] proposent des tissages dynamiques n'ayant pas l'inconvénient d'agir uniquement à posteriori, mais ils ne proposent que des points de jonction équivalents à l'appel de méthodes. Ces points de jonction ne permettent donc pas de détecter des séquences de messages. Notons tout de même qu'Aspect-J propose un mécanisme de *c-flow* qui permet d'identifier le contexte d'exécution d'une méthode. Avec ce mécanisme, on peut donc détecter la séquence d'appels d'une méthode (par exemple, savoir que c'est la méthode *a* qui a appelé la méthode *b* qui a appelé la méthode *c*). Mais cela ne permet pas de détecter des séquences de messages dans un cadre plus général.

Illustrons maintenant le processus de tissage proposé dans cette thèse. Le tissage de l'aspect comportemental de la figure 1.19 est décrit à travers l'exemple de la figure 1.20. Le bMSC de base (le bMSC dans lequel l'aspect va être tissé) décrit un comportement où un client essaie de s'identifier (*log in*) sur un serveur, mais ses deux premières tentatives échouent. Il y réussit à sa troisième tentative. La première étape du tissage consiste à détecter, en séquence, le comportement décrit par l'expression de coupe dans le bMSC *base*, c'est-à-dire le message *log in* suivi du message *réessaie*. Les comportements détectés sont les points de jonction, et ils sont représentés par des flèches en pointillés dans les parties grisées de la figure 1.20. Une fois les points de jonction obtenus, il reste à composer le comportement décrit par l' advice avec le bMSC de base au niveau des points de jonction. On obtient alors le bMSC de droite sur la figure 1.20. La phase de composition sera traitée en détail dans le chapitre 4 tandis le chapitre suivant détaillera la phase de détection dans des bMSCs.

Dans cette thèse, nous nous intéressons également au tissage dans des scénarios infinis. Le scénario de base est donc un HMSC (avec au moins une boucle). Le principe est le même que celui que l'on vient de présenter, le tissage étant effectué dans l'ensemble potentiellement infini de bMSCs que le HMSC de base génère. Le problème délicat est celui de la détection des points de jonction : comment détecter tous les points de jonction sans pour autant déplier l'ensemble potentiellement infini des comportements qu'un HMSC génère. Ce problème sera traité au chapitre 3.

1.5 Conclusion

L'état de l'art présenté sur la modélisation orientée-aspect nous a montré qu'il n'existait pas de réel tisseur d'aspects pour des modèles dynamiques. Cette constatation nous a poussés à proposer notre propre tisseur d'aspect pour des modèles dynamiques particuliers que sont les Message Sequence Charts (MSCs). Ce tisseur sera présenté dans le

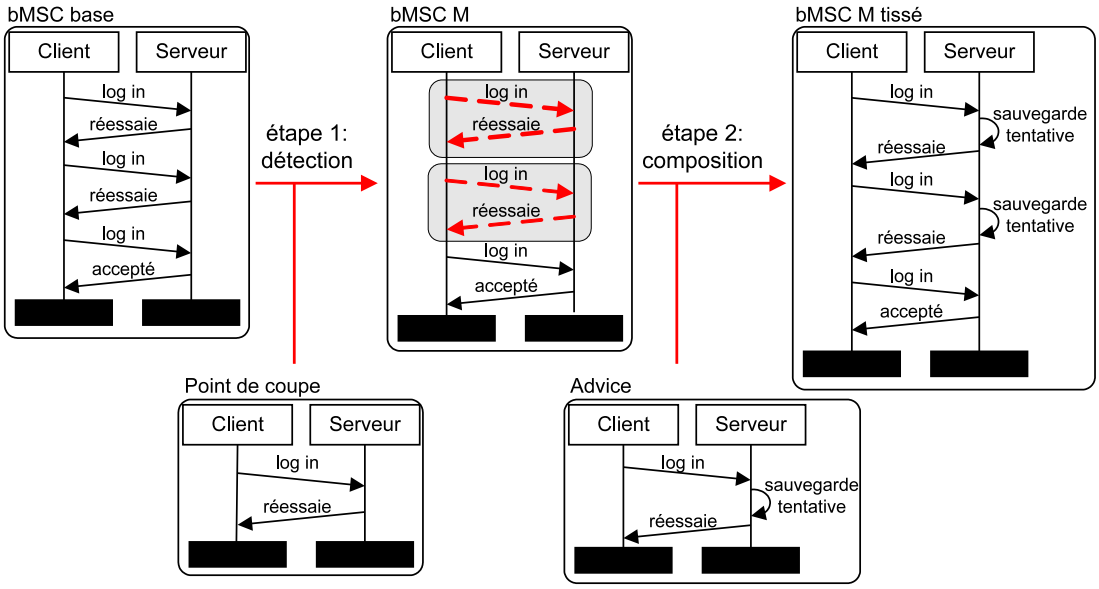


FIG. 1.20 – Illustration du processus de tissage

reste de ce document.

Dans ce chapitre, nous avons également défini la notion d'aspect comportemental comme une paire de bMSCs : un bMSC *expression de coupe* pour exprimer un comportement à rechercher et un bMSC *advice* pour spécifier le comportement à composer au niveau de chaque point de jonction. Ce type d'aspect permet de définir relativement facilement des expressions de coupe complexes telles que des successions de messages difficilement spécifiable par des langages de programmation orientés-aspect.

Chapitre 2

Processus de Détection dans des comportements finis

L'objectif de ce chapitre est de présenter un mécanisme de détection dans des comportements finis. La section 1 introduit différentes interprétations, définies formellement, de points de jonction dans des scénarios, facilitant plus ou moins le tissage d'aspects multiples. Un moyen d'ordonner des points de jonction successifs est également proposé. La section 2 présente des algorithmes qui permettent la détection de ces points de jonction dans un bMSC ou un ensemble fini de bMSCs. La section 3 présente un mécanisme de "caractères génériques"[1] sur les noms des messages qui augmente l'expressivité de notre langage d'expressions de coupe. Finalement, la section 4 conclut ce chapitre.

Notons que dans ce chapitre, nous ne considérerons que des bMSCs sans corégion, le comportement d'un bMSC avec corégion pouvant toujours être facilement représenté par le comportement de plusieurs bMSCs sans corégions. Notons également que conformément au chapitre précédent, nous définissons un aspect comportemental comme une paire de bMSCs $A = (P, Ad)$, où P est une expression de coupe et Ad un advice.

2.1 Différentes Définitions de Points de Jonction

Cette section a pour but de définir de manière précise la notion de point de jonction. Mais avant cela, nous allons expliquer pourquoi nous présentons plusieurs sémantiques de points de jonction. Une idée simple est de définir un point de jonction comme une sorte de "sous-bMSC", qui caractérise, entre autres, une séquence stricte de messages (une séquence de messages qui n'est entrelacée avec aucun autre comportement). Malheureusement, le tissage de plusieurs aspects au niveau d'un même point de jonction peut être difficile si le point de jonction est défini comme une stricte séquence de messages car des aspects tissés précédemment peuvent avoir inséré des messages entre les messages formant le point de jonction. Dans ce cas, le seul moyen de rendre possible le tissage de plusieurs aspects au niveau d'un même point de jonction, est de définir chaque aspect en fonction des autres aspects, ce qui est difficilement acceptable.

[1] wildcards en anglais

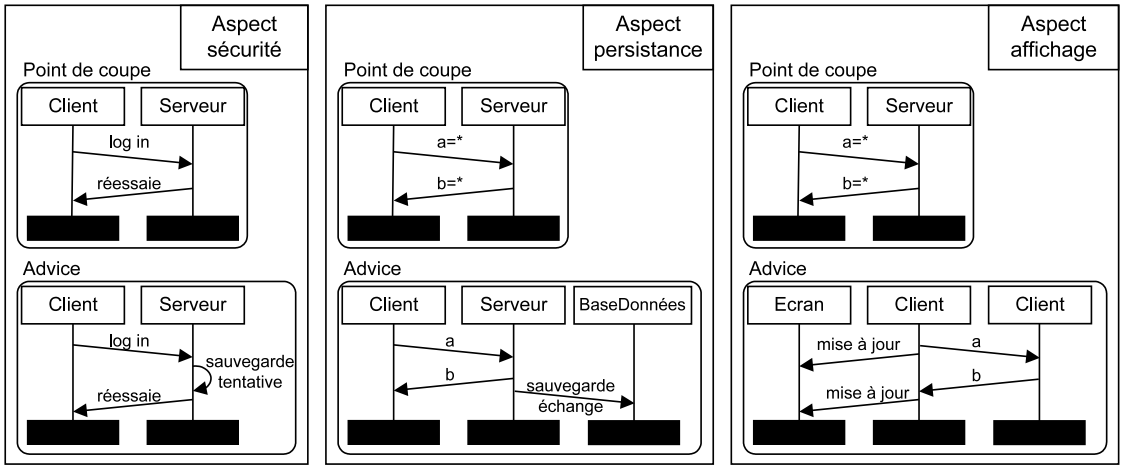


FIG. 2.1 – Trois aspects comportementaux

Pour mieux expliquer le problème, considérons les trois aspects décrits dans la figure 2.1, ainsi que le tissage de ces aspects dans le bMSC *base* de la figure 2.2. Le bMSC *base* représente les tentatives de "log in" d'un client sur un serveur. Nous qualifions le premier aspect comportemental de la figure 2.1 d'aspect de sécurité car il permet d'identifier les tentatives de "log in" qui échouent. Le deuxième aspect, qualifié d'aspect de persistance, permet de sauvegarder dans une base de données tous les échanges entre le client et le serveur. Dans l'expression de coupe de cet aspect, nous utilisons des caractères génériques, c'est-à-dire des expressions régulières sur les noms des messages. Sur cet exemple, le caractère "*" signifie que nous recherchons un message ayant un nom quelconque. Une explication détaillée du mécanisme de caractères génériques sera fournie dans la section 2.3. Enfin, le troisième aspect permet de mettre à jour un écran à chaque fois que le client envoie ou reçoit un message. La figure 2.2 représente le tissage attendu des trois aspects comportementaux dans le bMSC *base*. Or, si les points de jonction sont définis comme une stricte séquence de messages correspondant à ceux spécifiés par l'expression de coupe, le tissage des trois aspects de la figure 2.1 est impossible. En effet, quand l'aspect *sécurité* est tissé, un message *sauvegarde tentative* est ajouté entre les messages *log in* et *réessaie*. Puisque l'expression de coupe détecte uniquement des strictes séquences de messages, après le tissage de l'aspect de *sécurité*, l'aspect $affichage$ ne peut plus être tissé. Nous avons le même problème si nous tissons d'abord l'aspect $affichage$ et ensuite l'aspect *sécurité*.

Pour limiter ce problème de tissage multiple, qui est en réalité un problème non surprenant de confluence[2], nous introduisons plusieurs définitions formelles de la notion de points de jonction qui permettent la détection de points de jonction où des événements peuvent s'intercaler entre les événements spécifiés dans l'expression de coupe. De cette manière, quand l'aspect *sécurité* est tissé, l'expression de coupe de l'aspect $affichage$

[2]En effet, il n'est pas surprennant que le résultat d'un tissage d'un aspect A_1 suivi d'un aspect A_2, n'est pas le même si on tisse d'abord A_2 puis A_1.

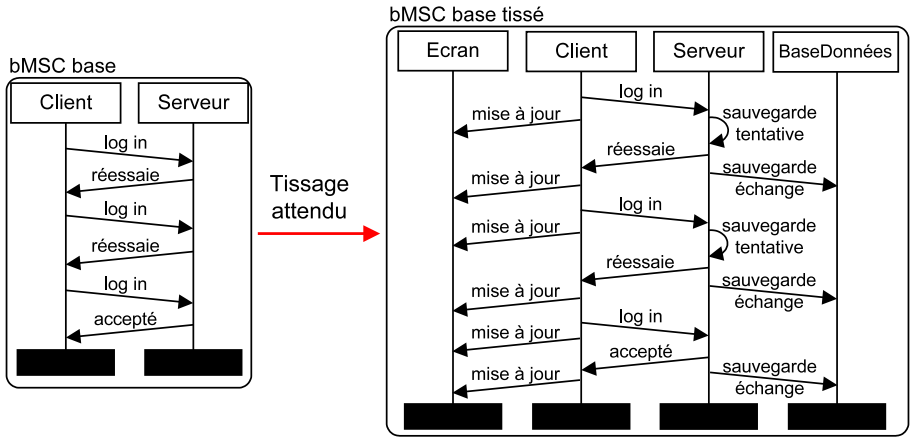


FIG. 2.2 – Tissage attendu des trois aspects de la figure 2.1 dans le bMSC M

permettra la détection des points de jonction formés par les messages *log in* et *réessaie* même si le message *sauvegarde tentative* a été ajouté.

2.1.1 Notions de parties d'un bMSC

La définition de point de jonction sera liée à la notion de *partie* d'un bMSC. Pour une expression de coupe P donnée, une partie M' d'un bMSC sera appelée point de jonction s'il existe un isomorphisme entre P et M'. Pour définir de manière plus rigoureuse un point de jonction, il est donc nécessaire de définir ce que l'on appelle partie d'un bMSC.

Nous proposons quatre définitions de parties d'un bMSC permettant la définition de quatre types de points de jonction différents. Ces parties sont appelées : *motif, motif sûr, motif clos et sous-bMSC*. La figure 2.3 permet de se faire une première idée intuitive de ce que ces parties peuvent représenter. En effet, les messages $m1$ et $m2$ (mis en pointillés) forment un *motif* dans les quatre bMSCs $M1$, $M2$, $M3$, et $M4$. Les deux messages forment un *motif sûr* dans les bMSCs $M1$, $M2$ et $M3$. Ils forment un *motif clos* dans les bMSCs $M1$ et $M2$, et un *sous-bMSC* uniquement dans le bMSC $M1$.

De manière plus formelle, voici la définition d'un motif d'un bMSC :

Définition 2.1 (Motif) *Soit* $M = (I, E, \leq, A, \alpha, \phi, \prec)$ *un bMSC. Nous dirons que* $M' = (I', E', \leq', A', \alpha', \phi', \prec')$ *est un motif de* M *si :*

- $I' \subseteq I, E' \subseteq E, A' \subseteq A, \alpha' = \alpha_{|E'}, \phi' = \phi_{|E'}$;
- $\leq' \subseteq \leq_{|E'}$,
- $\forall (e, f) \in \prec, e \in E' \Leftrightarrow f \in E'$, *la réception correspondante d'un envoi de message appartient au motif si l'envoi appartient au motif et inversement (un message ne peut donc pas être coupé).*

Un motif M' d'un bMSC M est une partie qui peut être croisée[3] par des messages. Dans la figure 2.3 les messages $m1$ et $m2$ forment un motif du bMSC $M2$ alors que

[3]pour rappel, le terme croisement est défini à la définition 1.4 du chapitre 1

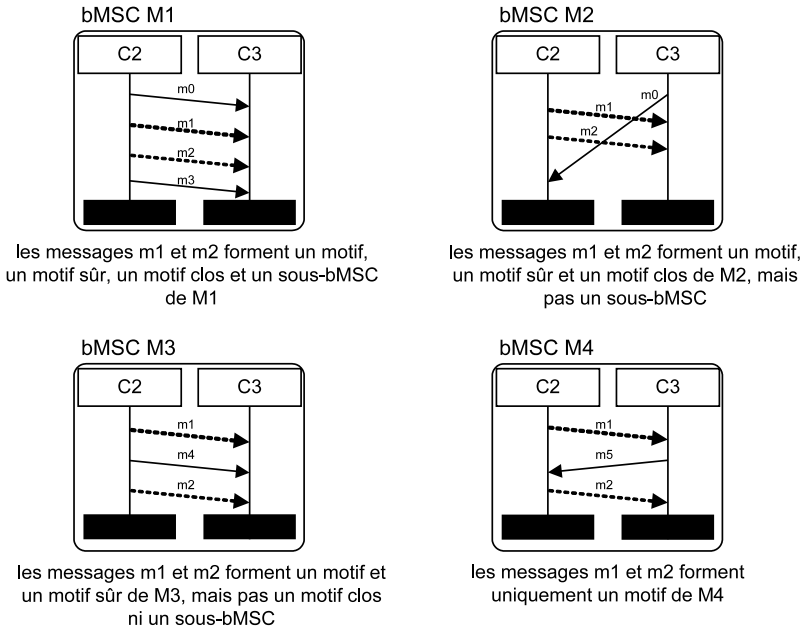


FIG. 2.3 – Illustration des notions de partie

le message $m0$ les croise. Un motif M' autorise également la présence de messages intercalés entre les messages de M'. C'est le cas dans les bMSCs $M3$ et $M4$ de la figure 2.3, où les messages $m1$ et $m2$ forment un motif alors que les messages $m4$ ou $m5$ sont présents entre les messages $m1$ et $m2$. Enfin, une caractéristique importante d'un motif M' d'un bMSC M est que l'ordre des événements défini par les événements de M', n'est pas forcément égal à l'ordre du bMSC M restreint aux événements de M', c'est-à-dire que $\leq'$ n'est pas forcément égal à $\leq_{|E'}$. Dans le bMSC $M4$ de la figure 2.3, les messages $m1$ et $m2$ forment un motif M' de $M4$ alors que $\leq' \neq \leq_{|E'}$[4]. En effet, dans $M4$, le message $m5$ impose un ordre entre la réception du message $m1$ et l'émission du message $m2$, alors que dans M' où il n'y a que les messages $m1$ et $m2$, la réception du message $m1$ et l'émission du message $m2$ sont concourrentes.

Cette caractéristique différencie un motif d'un motif sûr. Pour une partie M' d'un bMSC M, M' sera qualifiée de motif sûr si M' est un motif et que l'ordre défini par les événements de M' est égale à l'ordre des événements de M restreint aux événements de M'. Formellement, un motif sûr est défini par :

Définition 2.2 (Motif Sûr) *Soit $M = (I, E, \leq, A, \alpha, \phi, \prec)$ un bMSC. Nous dirons que $M' = (I', E', \leq', A', \alpha', \phi', \prec')$ est un motif sûr de M si :*

- *M' est un motif de M pour lequel $\leq' = \leq_{|E'}$.*

Dans la figure 2.3, les messages $m1$ et $m2$ forment un motif sûr dans les bMSCs $M1$, $M2$ et $M3$. Insistons sur le fait que dans un motif sûr, l'ordre des événements est le

[4] E' étant l'ensemble d'événements formé par les envois et réceptions des messages $m1$ et $m2$

même que celui du bMSC initial restreint aux événements du motif sûr. C'est pour cette raison que les messages $m1$ et $m2$ ne forment pas un motif sûr dans $M4$. Par contre, un motif sûr n'est pas forcément *clos*, c'est-à-dire que des événements peuvent être présents entre les événements du motif (par exemple, il y a un message $m4$ entre les messages $m1$ et $m2$ dans le bMSC $M3$).

La troisième définition proposée caractérise une partie qui peut être croisée par des messages non contenus dans cette partie, mais où aucun message ne peut s'intercaler entre les messages de cette partie. Plus formellement :

Définition 2.3 (Motif Clos) *Soit $M = (I, E, \leq, A, \alpha, \phi, \prec)$ un bMSC. Nous dirons que $M' = (I', E', \leq', A', \alpha', \phi', \prec')$ est un motif clos de M si :*
– M' est un motif sûr de M et $\widehat{\downarrow}_{\leq,E} E' \cap \widehat{\uparrow}_{\leq,E} E' = E'$.

La quantité $\widehat{\downarrow}_{\leq,E} E' \cap \widehat{\uparrow}_{\leq,E} E'$, qui représente l'intersection entre l'ensemble des prédécesseurs de E'[5] et l'ensemble des successeurs de E', permet de détecter si des événements sont intercalés entre l'ensemble E' des événements d'une partie M'. En effet, si un événement e est intercalé entre deux événements e' et e'' de M' ($e' \leq e \leq e''$ et $\phi(e') = \phi(e) = \phi(e'')$), alors e appartient à $\widehat{\downarrow}_{\leq,E} E'$ et à $\widehat{\uparrow}_{\leq,E} E'$. Donc $\widehat{\downarrow}_{\leq,E} E' \cap \widehat{\uparrow}_{\leq,E} E' \neq E'$

Dans la Figure 2.3, les messages $m1$ et $m2$ forment un motif clos dans les bMSCs $M1$ et $M2$. Nous notons qu'aucun événement n'appartenant pas au motif clos n'est intercalé entre les événements du motif clos. De plus, nous notons qu'un motif clos peut être croisé par d'autres événements (par ex., dans le bMSC $M2$, le motif clos est croisé par le message $m0$).

La dernière définition proposée est celle d'un sous-bMSC :

Définition 2.4 (Sous-bMSC) *Soit M un bMSC. Nous dirons que M' est un sous-bMSC de M s'il existe deux bMSCs X et Y tel que $M = X \bullet M' \bullet Y$ ($\bullet$ étant l'opérateur de composition séquentielle)*

Cette dernière définition est la plus restrictive parmi celles proposées. Dans la Figure 2.3, les messages $m1$ et $m2$ forment un sous-bMSC uniquement dans le bMSC $M1$. Un sous-bMSC M' ne peut pas être croisé par des messages (par exemple, dans le bMSC $M2$ de la figure 2.3, le message $m0$ croise les messages $m1$ et $m2$, qui ne forment donc pas un sous-bMSC de $M2$). De plus, la décomposition sous la forme $X \bullet M' \bullet Y$ impose qu'aucun événement ne puisse être présent "entre" des événements d'un sous-bMSC (dans les bMSCs $M3$ et $m4$ les messages $m1$ et $m2$ ne forment pas un sous-bMSC car les messages $M4$ et $M5$ sont intercalés entre $m1$ et $m2$).

Notons qu'un sous-bMSC permet de caractériser des séquences strictes de messages, mais pas uniquement. En effet, des éléments parallèles peuvent également former un sous-bMSC. Prenons par exemple le bMSC M de la figure 2.4. M peut être décomposé en $M = M_1 \bullet M_2 \bullet M_3$ comme détaillé sur la figure. M_2 est donc un sous-bMSC de M et pourtant ce n'est pas une séquence stricte de messages.

[5] notons que E' est forcément inclus dans $\widehat{\downarrow}_{\leq,E} E'$ et dans $\widehat{\uparrow}_{\leq,E} E'$ car tout événement de E' est son propre prédécesseur et son propre successeur ($e \leq e$)

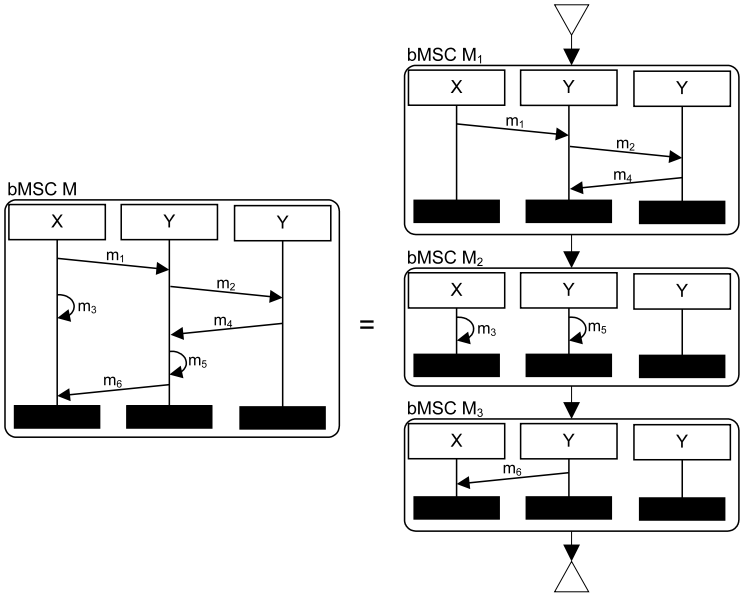


FIG. 2.4 – Illustration d'un sous-bMSC

2.1.2 Point de Jonction

Pour définir formellement la détection de points de jonction dans un scénario de base, il est nécessaire de définir la notion de morphisme et d'isomorphisme entre bMSCs.

Définition 2.5 (Morphisme de bMSCs) *Soit deux bMSCs* $M = (I, E, \leq, A, \alpha, \phi, \prec)$ *et* $M' = (I', E', \leq', A', \alpha', \phi', \prec')$. *Un morphisme de bMSCs, de* M *vers* M' *est un triplet* $\mu =< \mu_0, \mu_1, \mu_2 >$ *de morphismes, où* $\mu_0 : I \to I'$, $\mu_1 : E \to E'$ *est injective,* $\mu_2 : A \to A'$ *est une fonction de renomage et :*

$$(i)\ \ \forall (e,f) \in E^2, e \leq f \Rightarrow \mu_1(e) \leq' \mu_1(f) \qquad (iii)\ \ \mu_0 \circ \phi = \phi' \circ \mu_1$$
$$(ii)\ \ \forall (e,f) \in E^2, e \prec f \Rightarrow \mu_1(e) \prec' \mu_1(f) \qquad (iv)\ \ \mu_2 \circ \alpha = \alpha' \circ \mu_1$$

Notons que les propriétés (i) et (ii) garantissent que par un morphisme de bMSCs l'ordre des événements est préservé ainsi que la nature des événements (par exemple, un envoi de message de M sera toujours associé à un envoi de message de M'). La propriété (iii) signifie que tous les événements localisés sur un objet de M sont envoyés sur un seul objet de M'. La Figure 2.5 montre un morphisme $f =< f_0, f_1, f_2 >: P \to M2$ où seul le morphisme f_1 associant les événements est représenté (par ex., l'événement ep_1, qui représente l'envoi du message $m1$, est associé à l'événement em_2).

Définition 2.6 (Isomorphisme de bMSCs) *Un morphisme de bMSCs* $\mu = (\mu_0, \mu_1, \mu_2)$ *d'un bMSC* M *à un bMSC* M' *est un isomorphisme si les trois morphismes* μ_0, μ_1, *et* μ_2 *sont des isomorphismes.*

Avec cette définition d'isomorphisme entre bMSCs, nous pouvons définir de manière générale la notion de point de jonction dans un scénario :

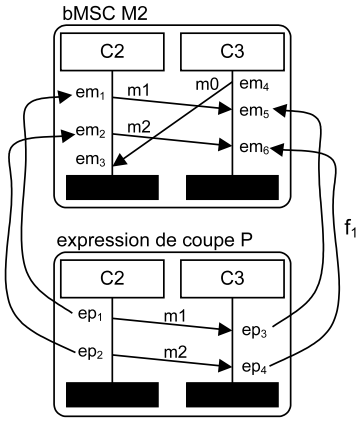


FIG. 2.5 – Illustration d'un morphisme de bMSCs

Définition 2.7 (point de jonction) *Soient M un bMSC et P une expression de coupe. Soit J_1 une partie de M. Nous dirons que J_1 est un point de jonction si et seulement si il existe un isomorphisme de bMSCs $\mu = (\mu_0, \mu_1, \mu_2)$ de P vers J_1 où les morphismes μ_0 et μ_2 sont des morphismes identités (P et J_1 ont les mêmes instances et les mêmes noms d'actions).*

2.1.3 Points de jonction successifs

Pour résumer la sous-section précédente, pour chaque définition de partie d'un bMSC, il y a une définition correspondante de point de jonction. Dans la figure 2.5, en considérant l' expression de coupe P décrite, il est facile de vérifier que les messages $m1$ et $m2$ forment un point de jonction correspondant à l'expression de coupe P si une partie est définie comme un motif, un motif sûr ou un motif clos, car il existe un isomorphisme de bMSCs entre P et un motif, un motif sûr ou un motif clos de $M2$.

Cependant, cette définition de point de jonction n'est pas assez précise. En effet, dans la figure 2.6, l'expression de coupe P_1 corresponds à deux différentes parties de M_1. Il y a donc deux points de jonction possibles J_1 et J_2, mais ces points de jonction sont entremêlés. Considérons maintenant l'expression de coupe P_2 et le bMSC M_2. Si nous prenons comme définition de partie la définition de motif, il y a quatre points de jonction possibles : J_1, J_2, J_3 et J_4. En effet, le premier message a et le premier message b forment le point de jonction J_1. Le second message a et le second message b forment le point de jonction J_2, alors que le premier message a combiné avec le second message b et le deuxième message a combiné avec le premier message b forment respectivement les points de jonction J_4 et J_3.

Parmi tous ces points de jonction conflictuels, lesquels choisir ? Pour résoudre ce problème, dans un premier temps nous proposons une manière d'ordonner des parties d'un bMSC isomorphes à une expression de coupe dont le but de déterminer le premier point de jonction correspondant à l'expression de coupe. Nous montrerons alors que le premier point de jonction est toujours unique, car l'ordre défini sur les parties est un

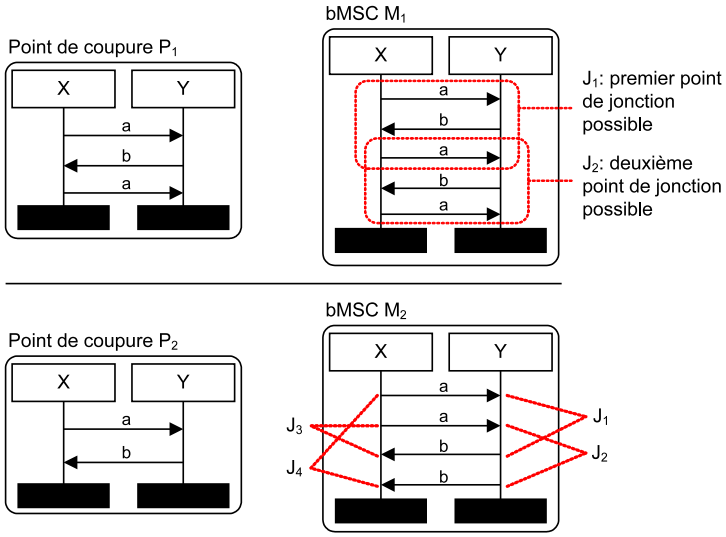


FIG. 2.6 – Multiples points de jonction possible

treillis. Dans un deuxième temps, nous définissons de manière inductive la notion de points de jonction successifs en considérant le premier point de jonction J apparaissant dans un bMSC M, et en continuant sur M moins J.

Définition 2.8 (Parties Ordonnées) *Soient $M = (I_M, E_M, \leq_M, A_M, \alpha_M, \phi_M, \prec_M)$ un bMSC et $P = (I_P, E_P, \leq_P, A_P, \alpha_P, \phi_P, \prec_P)$ une expression de coupe. Soient J_1 et J_2 deux parties de M tel qu'il existe deux isomorphismes de bMSCs $f = (f_0, f_1, f_2) : P \to J_1$ et $g = (g_0, g_1, g_2) : P \to J_2$. Nous dirons que J_1 précède J_2 (ou que J_2 succède J_1), noté $J_1 \ll J_2$, si et seulement si :*

$$\forall e \in E_P \text{ tel que } T(e) = envoi, f_1(e) \leq_M g_1(e)$$

Dans la figure 2.6, avec cette définition d'ordre, le point de jonction J_1 précède J_2 dans le bMSC M_1. Dans le bMSC M_2, nous constatons que J_1 précède tous les autres points de jonction.

Par la suite nous nous intéresserons particulièrement à la partie *minimum* de l'ordre $\ll$, c'est-à-dire la partie qui précède toutes les autres parties. Notons $\mathcal{J}_{P,M}$ l'ensemble de toutes les parties d'un bMSC M isomorphes à une expression de coupe P, c'est-à-dire tous les points de jonction relatifs à P et M. Nous notons également par $min(\mathcal{J}_{P,M})$ la partie minimale de $\mathcal{J}_{P,M}$, et nous appellerons *point de jonction minimum* cette partie. Cependant, $\ll$ ne définit pas un ordre total. Par exemple, dans la figure 2.6, J_3 et J_4 ne sont pas ordonnés par $\ll$. Il n'est donc pas évident que $min(\mathcal{J}_{P,M})$ soit unique. Pour prouver l'unicité de $min(\mathcal{J}_{P,M})$, nous montrons que l'ordre $\ll$ est un treillis.

Théorème 2.1 *Soient $\mathcal{J}_{P,M}$ l'ensemble des points de jonction d'un bMSC M correspondant à une expression de coupe P et $\ll$ l'ordre sur ces points de jonction comme défini par la définition 2.8, alors $(\mathcal{J}_{P,M}, \ll)$ est un treillis.*

Démonstration :

Pour démontrer le théorème précédent, nous utiliserons le lemme suivant que l'on peut trouver dans le livre *"Introduction to Lattices and Order"* [DP90] (p. 110) :

Lemme :

Soit un triplet $(L, \vee, \wedge)$ où L est un ensemble non vide et où $\vee$ et $\wedge$ sont deux opérateurs binaires qui satisfont pour tout $a, b, c \in L$:

- (1) $a \vee a = a$ et $a \wedge a = a$ (idempotence) ;
- (2) $a \vee b = b \vee a$ et $a \wedge b = b \wedge a$ (commutativité) ;
- (3) $(a \vee b) \vee c = a \vee (b \vee c)$ et $(a \wedge b) \wedge c = a \wedge (b \wedge c)$ (associativité) ;
- (4) $a \vee (a \wedge b) = a$ et $a \wedge (a \vee b) = a$ (absorption).

alors :

- (i) $\forall a, b \in L, a \vee b = b \Leftrightarrow a \wedge b = a$;
- (ii) Si on défini $\leq$ par $a \leq b$ si $a \vee b = b$, alors $\leq$ est une relation d'ordre ;
- (iii) Avec $\leq$ comme en (ii), $(L, \leq)$ est un treillis tel que $\forall a, b \in L, a \vee b = sup\{a, b\}$ et $a \wedge b = inf\{a, b\}$.

■

Nous allons prouver que $\mathcal{J}_{P,M}$ peut être munis de deux lois $\vee$ et $\wedge$ qui vérifient les propriétés 1 à 4 du lemme.

Soit $\mathcal{J}_{P,M}$ l'ensemble des points de jonction relatifs à une expression de coupe $P = (I_P, E_P, \leq_P, A_P, \alpha_P, \phi_P, \prec_P)$ et à un bMSC de base $M = (I, E, \leq, A, \alpha, \phi, \prec)$. Soient $\vee$ et $\wedge$ les opérateurs définis pour tout J_i, J_j de $\mathcal{J}_{P,M}$ par :

$$\begin{aligned} J_i \vee J_j = \quad & \{e, f \in J_i | e \prec f, \exists e' \in E_P, e = \mu_{i_1}(e'), \mu_{j_1}(e') \leq e\} \cup \\ & \{e, f \in J_j | e \prec f, \exists e' \in E_P, e = \mu_{j_1}(e'), \mu_{i_1}(e') \leq e\} \end{aligned}$$

$$\begin{aligned} J_i \wedge J_j = \quad & \{e, f \in J_i | e \prec f, \exists e' \in E_P, e = \mu_{i_1}(e'), e \leq \mu_{j_1}(e')\} \cup \\ & \{e, f \in J_j | e \prec f, \exists e' \in E_P, e = \mu_{j_1}(e'), e \leq \mu_{i_1}(e')\} \end{aligned}$$

$\mu_i =< \mu_{i_0}, \mu_{i_1}, \mu_{i_2} >$ et $\mu_j =< \mu_{j_0}, \mu_{j_1}, \mu_{j_2} >$ étant les isomorphismes liant P aux points de jonction respectifs J_i et J_j.

Pour $(\mathcal{J}_{P,M}, \vee, \wedge)$, les propriétés (1) et (2) du lemme sont vérifiés (trivial). De même, soient J_i, J_j et J_k trois points de jonction et $\mu_i =< \mu_{i_0}, \mu_{i_1}, \mu_{i_2} >$, $\mu_j =< \mu_{j_0}, \mu_{j_1}, \mu_{j_2} >$ et $\mu_k =< \mu_{k_0}, \mu_{k_1}, \mu_{k_2} >$ les trois isomorphismes associant respectivement P à J_i, J_j et J_k. Soit deux événements e et f de M tel que $e \prec f$. Si e et f appartiennent à J_i et à $(J_i \vee J_j) \vee J_k$, soit e' l'événement correspondant dans P tel que $e = \mu_{i_1}(e')$, alors d'après la définition de $\vee$, e succède $\mu_{j_1}(e')$ et $\mu_{k_1}(e')$. Donc, e et f appartiennent également à $J_i \vee (J_j \vee J_k)$. On montre ainsi facilement que $(J_i \vee J_j) \vee J_k = J_i \vee (J_j \vee J_k)$. Par le même procédé, on montre également que $(J_i \wedge J_j) \wedge J_k = J_i \wedge (J_j \wedge J_k)$. Enfin, pour prouver la propriété (4), considérons deux points de jonction J_i et J_j et leurs morphismes associés μ_i et μ_j. Soient deux événements e_2 et f_2 appartenant à J_j et à $J_i \vee (J_i \wedge J_j)$ (et donc à $J_i \wedge J_j$) mais pas à J_i. Notons e' l'événement appartenant à P tel que $e_2 = \mu_{j_1}(e')$. Soit $e_1 = \mu_{i_1}(e')$, alors comme e_2 appartient à $J_i \wedge J_j$, $e_2 \leq e_1$, et comme e_2 appartient à $J_i \vee (J_i \wedge J_j)$, $e_1 \leq e_2$. Impossible, donc tous les événements de $J_i \vee (J_i \wedge J_j)$ appartiennent à J_i.

D'après le lemme, $(\mathcal{J}_{P,M}, \ll')$, avec $\ll'$ défini par $J_i \ll' J_j$ si $J_i \vee J_j = J_j$, est donc un treillis. Or $\ll'$ est équivalent à l'ordre $\ll$ de la définition 2.8. L'équivalence est facile à démontrer. Soient J_i et J_j deux points de jonction, et μ_i et μ_j leurs isomorphismes vers P associés. Si $J_i \ll' J_j$, par définition $J_i \vee J_j = J_j$, et donc tous les événements correspondant à un envoi de messages de J_j succèdent ceux de J_i. La réciproque est triviale.

⊡

Pour illustration, la figure 2.7 montre le diagramme de Hasse de l'ordre sur les points de jonction présents dans le bMSC $M2$ de la figure 2.6. Ce diagramme a bien la forme d'un treillis.

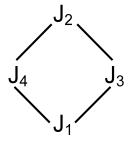


FIG. 2.7 – Diagramme de Hasse de l'ordre sur les points de jonction présent dans le bMSC $M2$ de la figure 2.6

Nous pouvons maintenant définir de manière inductive la notion de points de jonction successifs par

Définition 2.9 (points de jonction successifs) *Soient M un bMSC et P une expression de coupe. Soient k parties $\{J_i\}_{i\in\{1..k\}}$ de M isomorphes à P. Ces k parties sont des points de jonction successifs de P si :*

- *J_1 est le point de jonction minimum correspondant à P dans M, c'est-à-dire le minimum de $(\mathcal{J}_{P,M}, \ll)$;*
- *$\forall i \in \{2 \dots k\}$, J_i est le point de jonction minimum correspondant à P dans M', M' étant le bMSC qui contient tous les événements de M moins les événements de J_{i-1} et moins tous les événements qui précédent les événements de J_{i-1}, c'est-à-dire $M' = (M - \widehat{\downarrow}_{\leq,M} J_{i-1})$.*

Le fait de prendre à chaque fois le point de jonction minimum garantit l'unicité des points de jonction successifs. Notons cependant que le résultat $M' = (M - \widehat{\downarrow}_{\leq,M} J_{i-1})$ n'est pas toujours un bMSC clos pour les communications (définiton 1.7). En effet, dans la figure 2.8, le point de jonction minimum J_1 dans M correspondant à P, est constitué par les deux premiers messages a et b. Lorsque l'on supprime les événements $\widehat{\downarrow}_{\leq,M} J_1$ (les événements de J_1 et ceux qui les précédent), nous supprimons l'événement correspondant à l'envoi du message c. Donc, le résultat $M' = M - \widehat{\downarrow}_{\leq,M} J_1$ est formé par les deux derniers messages a et b et uniquement la réception du message c. En pratique, ces bMSCs non clos pour les communications ne sont pas réellement un problème, car les algorithmes proposés par la suite pourront être appliqués à des bMSCs de cette sorte.

2.1.4 Quelles définitions de parties choisir ?

Nous n'allons pas répondre directement à cette question. Nous allons simplement énumérer des arguments pour ou contre certaines définitions. Auparavant, il est impor-

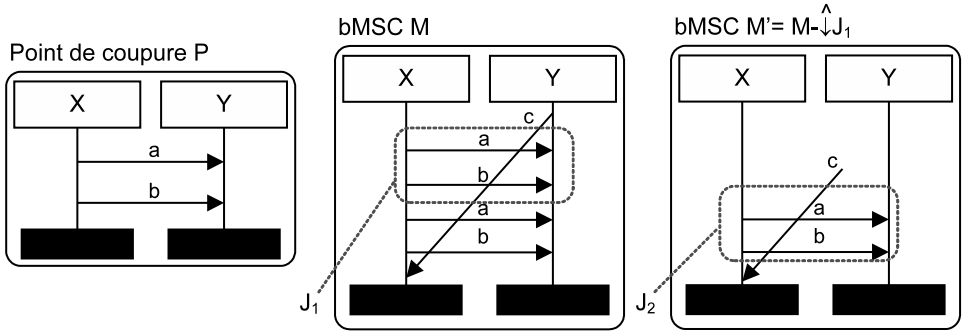


FIG. 2.8 – Exemple d'un bMSC non clos pour les communications

tant de noter que pour les quatre définitions d'une partie d'un bMSC proposées, les définitions sont fondées sur la sémantique du langage de scénarios utilisé puisque l'on ne considère pas seulement le nom des messages, mais également l'ordre partiel défini par l'expression de coupe (à l'exception du motif).

La définition d'un sous-bMSC est la plus *restrictive*, car avec cette définition, le comportement recherché peut être présent dans un bMSC, sans pour autant être détecté, s'il est, par exemple, croisé par un message. D'un autre côté, cette définition reste *simple* : on recherche uniquement une décomposition du bMSC de base M de la forme $M = M_1 \bullet J \bullet M_2$. Comme nous allons le montrer dans le chapitre suivant, cette simplicité permet d'obtenir des résultats intéressants lorsque l'on parle de détection d'un point de jonction dans un comportement infini.

La définition d'un motif est, à l'opposé, la moins contraignante. Des messages peuvent être intercalés entre le comportement recherché, ce qui peut entraîner la détection de points de jonction entremêlés avec des comportements non prévus par l'utilisateur. De plus, l'ordre partiel défini par l'expression de coupe n'est pas forcément préservé dans un point de jonction détecté. L'avantage d'un point de jonction défini comme un motif reste la faculté de tisser facilement plusieurs aspects au niveau d'un même point de jonction.

Les définitions de motif clos et de motif sûr combinent les avantages et inconvénients d'un sous-bMSC et d'un motif. Un motif clos "ressemble" à un sous-bMSC, mais il peut être croisé par des messages. Donc, si l'on désire rechercher de manière sûr une séquence stricte de messages, cette définition semble bien adaptée. Un motif clos garde l'inconvénient de ne pas bien supporter le tissage de plusieurs aspects au niveau d'un même point de jonction. Si l'on veut tisser plusieurs aspects au niveau d'un même point de jonction, tout en étant sûr que l'ordre partiel défini par l'expression de coupe est respecté, la définition de motif sûr semble bien adaptée. Un motif sûr gardant l'inconvénient de la détection de points de jonction entremêlés avec des comportements non prévus par l'utilisateur, car des messages peuvent s'intercaler entre les messages des points de jonction.

Malgré cette courte discussion sur les avantages et inconvénients de chaque définition, l'intérêt de l'approche proposée est qu'un utilisateur puisse choisir à sa guise la

sémantique voulue pour son tissage dans des comportements finis : il est donc libre de choisir la définition de parties qui lui convient le mieux, en adaptant l'algorithme de détection en fonction de la définition choisie. Nous montrerons comment cette flexibilité peut facilement être implantée dans l'environnement Kermeta dans le chapitre 5.

2.2 Algorithmes de détection de points de jonction dans un bMSC

Nous proposons quatre algorithmes différents, un pour chaque définition de points de jonction proposée (motif clos, motif sûr, motif, sous-bMSC).

Par soucis de clarté, l'ordre de présentation des algorithmes ne suivra pas l'ordre des définitions de parties proposées dans la section précédente. L'algorithme de détection d'un motif clos sera présenté en premier, suivi par ceux permettant la détection d'un motif sûr et d'un motif (ces deux algorithmes ne présentent que quelques adjonctions et différences par rapport à l'algorithme de détection d'un motif clos). Enfin, nous terminerons par l'algorithme permettant la détection d'un sous-bMSC.

Pour chaque algorithme, nous noterons par $\pi_i(M) \subseteq E_M$ la projection d'un bMSC M sur un objet i de M et par $\pi_E(M)$ la restriction d'un bMSC M à un sous-ensemble $E \subseteq E_M$. De plus, nous utiliserons une fonction β_E qui pour un événement e de E donne la position de e sur l'objet contenant e. Plus précisément, la position d'un événement sur un objet est définie par le nombre d'événements qui le précèdent sur cet objet : $\forall e \in E, \beta_E(e) = |\{e' \in E | \phi(e) = \phi(e') \wedge e' \leq e\}|$. Enfin, la fonction $\Gamma_{E,o}(n)$ donne l'événement de E situé en n^{ime} position sur l'objet o ($\Gamma_{E,o}(n) = e$ si et seulement si $\beta_E(e) = n \wedge \phi(e) = o$).

Notons que puisqu'un bMSC est toujours fini (il ne contient qu'un ensemble fini d'événements), un algorithme permettant la détection de points de jonction ne pose pas de problème de terminaison.

Notons finalement que les algorithmes de détection dans un bMSC présentés dans cette section peuvent être appliqués à un HMSC sans cycle. En effet, un HMSC sans cycle peut être considéré comme un générateur d'un ensemble fini de bMSCs (voir la sous-section sur les HMSCs du chapitre précédent). Les algorithmes de cette section peuvent donc très facilement être adaptés pour prendre en entrée un HMSC sans cycle au lieu d'un bMSC. Pour cela, il suffit d'appliquer les algorithmes à chaque bMSC généré.

2.2.1 Algorithme de détection d'un motif clos

L'algorithme 1 permet de construire un isomorphisme $\mu = (\mu_0, \mu_1, \mu_2)$ d'une expression de coupe P vers le premier motif clos M' d'un bMSC M, tel que μ_0 et μ_2 soient des morphismes identités (donc M' représente un point de jonction correspondant à P).

Pour une expression de coupe P et un bMSC M, la première partie de l'algorithme (ligne 1 à 4) trouve pour chaque instance i de P, une séquence d'événements de M situé sur i tel que les noms d'action (également appelés les noms de message) et les types

associés à ces événements sont exactement les noms d'action et les types des événements de l'ensemble formé par la restriction de P aux événements situés sur i. En d'autres mots, P est détecté localement sur l'instance i par chaque séquence extraite de M. La seconde partie de l'algorithme vérifie si toutes les séquences locales (instance par instance) peuvent être combinées et former une partie M' pour que P soit isomorphe à M'. Pour cela, on vérifie que pour chaque couple d'événements envoi-réception de P, les événements de M' à la même position forment le même couple envoi-réception. Si P a bien une correspondance dans M, l'ensemble des événements formant le point de jonction est construit en considérant la solution minimale de séquences locales, la notion de minimale ou de minimum étant celle introduite dans la sous-section 2.1.3 définissant la notion de points de jonction successifs. Finalement, l'isomorphisme $\mu = (\mu_0, \mu_1, \mu_2)$ est construit. En particulier, μ_1 qui est l'isomorphisme de E_P vers $E_{M'}$ est construit en associant, pour chaque objet o de I_p, l'événement de o en ième position, l'événement appartenant à $E_{M'}$ sur o en ième position.

Algorithme 1 Détection d'un motif clos : (P,M)

entrée : expression de coupe $P = (I_P, E_P, \leq_P, , A_P, \alpha_P, \phi_P, \prec_P)$,
bMSC $M = (I_M, E_M, \leq_M, A_M, \alpha_M, \phi_M, \prec_M)$

sortie : $\mu = (\mu_0, \mu_1, \mu_2) : P \to M'$, $M' = (I_{M'}, E_{M'}, \leq_{M'}, A_{M'}, \alpha_{M'}, \phi_{M'}, \prec_{M'})$ étant une partie close de M

1: Pour tout $i \in I_P$ faire
2: $w_i = \pi_i(M)$ `/* un mot de tous les événements de l'objet i */`
3: $V_i = \{v \in E^* \mid \exists u, w, w_i = u.v.w \wedge \alpha(v) = \alpha(\pi_i(P)) \wedge T(v) = T(\pi_i(P))\}$ `/*` $\forall v \in V_i$, `les événements de` v `ont le même nom d'action et le même type que les événements de` P `sur l'instance` i`*/`
4: Fin Pour
5: $E_{M'} = min\{v_1, \ldots, v_{|I_P|} \in V_1 \times \cdots \times V_{|I_P|} | \forall (e,f) \in \prec_P, (\Gamma_{v_{\phi(e)},\phi(e)} \circ \beta_{E_P}(e), \Gamma_{v_{\phi(f)},\phi(f)} \circ \beta_{E_P}(f)) \in \prec_M\}$
`/* on vérifie qu'un couple d'événements envoi-reception a bien une correspondance avec un couple envoi-reception dans` M`, et avec la fonction` min `on prend le premier ensemble` $v_1, \ldots, v_{|I_P|}$ `satisfaisant les propriétés*/`
6: Si $(E_{M'} = \emptyset)$ alors
7: retour(nul)
8: Sinon
9: μ_0 est l'isomorphisme identité de I_P vers $\phi_M(E_{M'})$,
10: μ_2 est l'isomorphisme identité de A_P vers $\alpha_M(E_{M'})$,
11: μ_1 est l'isomorphisme de E_P vers $E_{M'}$ tel que $\forall e \in E_P$, $\mu_1(e) = \Gamma_{v_{\phi(e)},\phi(e)} \circ \beta_{E_P}(e)$ `/*` μ_1 `est construit en associant, pour chaque instance` o `de` I_p`, à l'événement de` o `en ième position, l'événement appartenant à` $E_{M'}$ `sur` o `en ième position.*/`
12: retour($\mu = (\mu_0, \mu_1, \mu_2)$)
13: Fin Si

2.2.2 Algorithme de détection d'un motif sûr

Algorithme 2 Détection d'un motif sûr : (P,M)

entrée : expression de coupe $P = (I_P, E_P, \leq_P, , A_P, \alpha_P, \phi_P, \prec_P)$,
bMSC $M = (I_M, E_M, \leq_M, A_M, \alpha_M, \phi_M, \prec_M)$

sortie : $\mu = (\mu_0, \mu_1, \mu_2) : P \rightarrow M'$, $M' = (I_{M'}, E_{M'}, \leq_{M'}, A_{M'}, \alpha_{M'}, \phi_{M'}, \prec_{M'})$ étant un motif sûr de M

1: Pour tout $i \in I_P$ faire
2: $w_i = \pi_i(M)$ /* un mot de tous les événements de l'objet i */
3: $V_i = \{v = x_1.x_2...x_k \in E_M^* \mid \exists u_i \in E_M^*, i \in \{1 \dots k+1\}, w_i = u_1.x_1.u_2.x_2...u_k.x_k.u_{k+1} \wedge \alpha(v) = \alpha(\pi_i(P)) \wedge T(v) = T(\pi_i(P))\}$ /* $\forall v \in V_i$, les événements de v ont le même nom d'action et le même type que les événements de P sur l'instance i*/
4: Fin Pour
5: $E_{M'} = min\{v_1, \dots, v_{|I_P|} \in V_1 \times \dots \times V_{|I_P|} | \forall (e,f) \in \prec_P, \big(\Gamma_{v_{\phi(e)},\phi(e)} \circ \beta_{E_P}(e), \Gamma_{v_{\phi(f)},\phi(f)} \circ \beta_{E_P}(f)\big) \in \prec_M \wedge \leq_{M'} = \leq_{M|E_{M'}}\}$
/* on vérifie qu'un couple d'événements envoi-réception a bien une correspondance avec un couple envoi-réception dans M. On vérifie également que l'ordre des événements de $E_{M'}$ est le même que celui des événements de E_M restreint aux événements de $E_{M'}$ et avec la fonction min on prend le premier ensemble $v_1, \dots, v_{|I_P|}$ satisfaisant les propriétés*/
6: Si $E_{M'} = \emptyset$ alors
7: retour($null$)
8: Sinon
9: μ_0 est l'isomorphisme identité de I_P vers $\phi_M(E_{M'})$,
10: μ_2 est l'isomorphisme identité de A_P vers $\alpha_M(E_{M'})$,
11: μ_1 est l'isomorphisme de E_P vers $E_{M'}$ tel que $\forall e \in E_P$, $\mu_1(e) = \Gamma_{v_{\phi(e)},\phi(e)} \circ \beta_{E_P}(e)$ /* μ_1 est construit en associant, pour chaque instance o de I_p, à l'événement de o en ième position, l'événement appartenant à $E_{M'}$ sur o en ième position.*/
12: retour($\mu = (\mu_0, \mu_1, \mu_2)$)
13: Fin Si

L'algorithme permettant de détecter des motifs sûrs dans un bMSC (Algorithme 2) est similaire à l'algorithme 1, excepté pour deux points. Premièrement, la ligne 3 est remplacée par la ligne : $V_i = \{v = x_1.x_2...x_k \in E_M^* \mid \exists u_i, i \in \{1...k+1\}, w_i = u_1.x_1.u_2.x_2...u_k.x_k.u_{k+1} \wedge \alpha(v) = \alpha(\pi_i(P)) \wedge T(v) = T(\pi_i(P))\}$. De cette manière, nous pouvons détecter un point de jonction même si des événements (représentés par les u_i) sont présents entre les événements de ce point de jonction (il est clair que les u_i peuvent être nuls). Deuxièmement, à la ligne 5, nous devons également vérifier si l'ordre des événements de $E_{M'}$ est le même que celui des événements de E_M restreint aux

événements de $E_{M'}$ (ce qui revient à vérifier si $\leq_{M'}=\leq_{M|E_{M'}}$). Cette vérification doit être faite, car la présence d'événements autres que ceux du motif, peut introduire une différence entre $\leq_{M'}$ et $\leq_{M|E_{M'}}$, comme montré dans la section précédente.

2.2.3 Algorithme de détection d'un motif

Cet algorithme est très similaire à l'algorithme 2 permettant la détection d'un motif sûr, à l'exception d'un point : il n'est pas nécessaire de vérifier que l'ordre des événements de $E_{M'}$ est le même que celui des événements de E_M restreint aux événements de $E_{M'}$ (il n'est pas nécessaire de vérifier que $\leq_{M'}=\leq_{M|E_{M'}}$). Vu la ressemblance avec l'algorithme 2, l'algorithme n'est pas écrit.

2.2.4 Algorithme de détection d'un sous-bMSC

L'algorithme 3 permet la détection d'un sous-bMSC M' d'un bMSC M correspondant à une expression de coupe P. L'algorithme ressemble à l'algorithme 1 qui permet la détection d'un motif clos. La seule différence est qu'il faut vérifier qu'il existe deux bMSCs X_1 et X_2 tel que M peut être écrit sous la forme $M = X_1 \bullet M' \bullet X_2$. Pour cela, on peut vérifier qu'il existe une coupe c valide avant M', et une coupe valide après M'. (Pour rappel, une coupe valide est une coupe qui ne "coupe" pas de message. Ces notions ont été introduites dans [HLM00]. Elles sont également présentées dans le chapitre 1)

Algorithme 3 Détection d'un sous-bMSC : (P,M)

entrée : expression de coupe $P = (I_P, E_P, \leq_P, , A_P, \alpha_P, \phi_P, \prec_P)$,
bMSC $M = (I_M, E_M, \leq_M, A_M, \alpha_M, \phi_M, \prec_M)$

sortie : $\mu = (\mu_0, \mu_1, \mu_2) : P \to M'$, $M' = (I_{M'}, E_{M'}, \leq_{M'}, A_{M'}, \alpha_{M'}, \phi_{M'}, \prec_{M'})$ étant un sous-bMSC de M

1: Pour tout $i \in I_P$ faire
2: $w_i = \pi_i(M)$ /* un mot de tous les événements de l'objet i */
3: $V_i = \{v \in E^* \mid \exists u, w, w_i = u.v.w \wedge \alpha(v) = \alpha(\pi_i(P)) \wedge T(v) = T(\pi_i(P))\}$ /* $\forall v \in V_i$, les événements de v ont le même nom d'action et le même type que les événements de P sur l'instance i*/
4: Fin Pour
5: $E_{M'} = min\{v_1, \ldots, v_{|I_P|} \in V_1 \times \cdots \times V_{|I_P|} |$
$\forall (e, f) \in \prec_P, \big(\Gamma_{v_{\phi(e)},\phi(e)} \circ \beta_{E_P}(e), \Gamma_{v_{\phi(f)},\phi(f)} \circ \beta_{E_P}(f)\big) \in \prec_M \wedge$
$\exists X_1, X_2, M = X_1 \bullet M' \bullet X_2\}$
/* on vérifie qu'un couple d'événements envoi-reception a bien une correspondance avec un couple envoi-reception dans M, et on vérifie que M peut être décomposé en $X_1 \bullet M' \bullet X_2$. Avec la fonction min, on prend le premier ensemble $v_1, \ldots, v_{|I_P|}$ satisfaisant les propriétés*/
6: Si $E_{M'} = \emptyset$ alors
7: retour($null$)
8: Sinon
9: μ_0 est l'isomorphisme identité de I_P vers $\phi_M(E_{M'})$,
10: μ_2 est l'isomorphisme identité de A_P vers $\alpha_M(E_{M'})$,
11: μ_1 est l'isomorphisme de E_P vers $E_{M'}$ tel que $\forall e \in E_P$, $\mu_1(e) = \Gamma_{v_{\phi(e)},\phi(e)} \circ \beta_{E_P}(e)$ /* μ_1 est construit en associant, pour chaque instance o de I_p, à l'événement de o en ième position, l'événement appartenant à $E_{M'}$ sur o en ième position.*/
12: retour($\mu = (\mu_0, \mu_1, \mu_2)$)
13: Fin Si

2.3 Mécanisme de Caractères Génériques

Tout comme dans Aspect-J, il est très utile d'améliorer l'expressivité de notre langage d'expression de coupe en ajoutant un mécanisme de caractères génériques. Pour cela, nous proposons d'ajouter des caractères génériques sur les noms des actions (noms des messages) de l'expression de coupe. Nous allons alors sensiblement modifier les définitions d'une expression de coupe et d'un advice proposés jusqu'ici. Premièrement, une expression de coupe $P = (I_P, E_P, \leq_P, A_P, \alpha_P, \phi_P, V_P)$ est maintenant étendue par un ensemble de variables V_P, où chaque variable $v_i \in V_P$ est associée à une expression régulière R_{v_i}. De plus, au lieu d'associer uniquement une action à un événement, le morphisme α_P peut également associer une variable à un événement, c'est-à-dire $\alpha_P : E_P \to A_P \cup V_P$. Dans la figure 2.9, l'expression de coupe P contient une variable

m associée à l'expression régulière $*$, qui signifie que P permet la détection de points de jonction où le message *log in* d'un *client* vers un *serveur* est suivi par un message quelconque d'un *serveur* vers un *client*. De cette manière, si l'on désire tisser l'aspect $A1$ de la figure 2.9 dans le HMSC *log* de la même figure, P permet la détection de deux points de jonction : le message *log in* suivi par soit le message *accepté*, soit par le message *rejeté*.

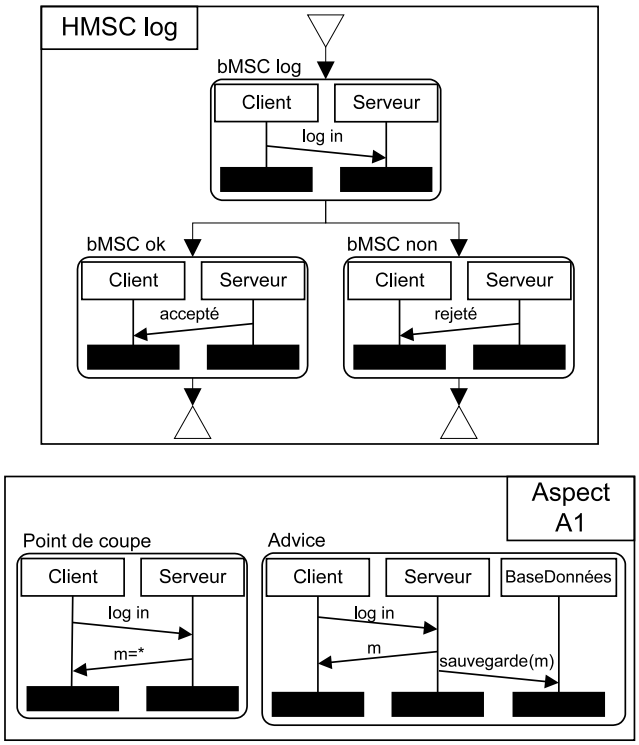


FIG. 2.9 – Exemple d'utilisation de caractères génériques

L'advice $Ad = (I_{Ad}, E_{Ad}, \leq_{Ad}, A_{Ad}, \alpha_{Ad}, \phi_{Ad}, V_{Ad} = V_P, \mathcal{F}_{Ad})$ est également étendu. Ad contient le même ensemble de variables que l'expression de coupe. De plus, Ad contient un ensemble de fonctions $\mathcal{F}_{Ad} : V'_{Ad} \subseteq V_{Ad} \rightarrow \Sigma^*$, qui permet de manipuler les variables de V_{Ad}. Par exemple, dans la figure 2.9, la fonction *sauvegarde* permet de prendre en paramètre la variable m correspondant à l'expression régulière $*$. Le résultat du tissage est représenté sur la figure 2.10 : la variable m est remplacée par les messages *accepté* et *rejeté*. Le morphisme α_{Ad} est également modifié. Il associe à un événement soit une action, soit une variable ou soit une fonction, c'est-à-dire α_{Ad} : $E_{Ad} \rightarrow A_{Ad} \cup V_{Ad} \cup \mathcal{F}_{Ad}$.

Avec cette notion d'expression de coupe et advice étendus, la définition de point de jonction est légèrement modifiée :

Définition 2.10 (point de jonction avec des caractères génériques)
Soient $M = (I_M, E_M, \leq_M, A_M, \alpha_M, \phi_M, \prec_M)$ *un bMSC et* $P = (I_P, E_P, \leq_P, A_P, \alpha_P,$

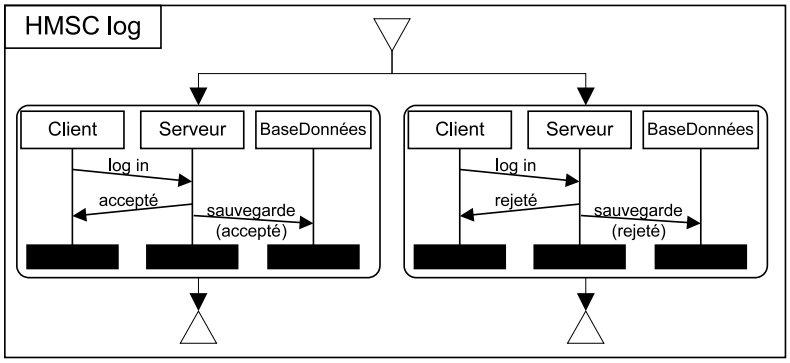


FIG. 2.10 – Résultat du tissage

ϕ_P, V_P) une expression de coupe avec caractères génériques. Soit J_1 une partie de M. Nous dirons que J_1 est un point de jonction si et seulement si

- *il existe un isomorphisme μ'_2 de $A_P \cup V_P$ vers A_{P_1} tel que $\forall x \in A_P, x = \mu'_2(x)$ et $\forall x \in V_P$, $\mu'_2(x)$ est conforme à l'expression régulière R_x associée à x ;*
- *il existe un isomorphisme $\mu = (\mu_0, \mu_1, \mu_2)$ de P vers P_1 tel que μ_0 et μ_2 sont des morphismes identités si $\forall x \in V_P$, x est remplacé par $\mu'_2(x)$ dans P.*

Les algorithmes de la section précédente subissent également une très légère modification au niveau de la ligne 3 qui permet de calculer l'ensemble des événements d'une instance ayant les mêmes noms d'action et le même type que ceux de P sur l'instance associée. Lors de ce calcul, il suffit de prendre en compte les expressions régulières présentes dans P pour que les algorithmes restent valides lorsque l'on utilise des caractères génériques.

Ce mécanisme de caractères génériques pourra également être utilisé pour compléter le mécanisme de détection de points de jonction dans des comportements infinis présenté dans le chapitre suivant.

2.4 Conclusion et Perspectives

Dans ce chapitre, nous avons présenté un mécanisme de détection de points de jonction correspondants à la description d'une expression de coupe. Nous avons présenté plusieurs sémantiques de points de jonction, ayant chacune des avantages et des inconvénients et facilitant plus ou moins le tissage de plusieurs aspects. Dans tous les cas, la détection proposée tient compte de l'ordre partiel induit par les bMSCs. Les bMSCs (ou les HMSCs sans cycle) ne définissant que des comportements finis, l'intérêt de ce travail porte réellement sur la proposition de plusieurs sémantiques de points de jonction, ainsi que sur le moyen d'ordonner des points de jonction successifs.

Comme travaux futurs, nous retiendrons deux extensions qui nous semblent intéressantes. Premièrement, il serait utile de proposer des conditions sur les aspects comportementaux, permettant de dire, par exemple, si deux aspects commutent, c'est-à-dire s'ils peuvent être tissés dans n'importe quel ordre, ou s'ils sont en conflit. Dans, ce dernier

cas, il est nécessaire de spécifier un ordre de tissage de ces aspects, ou bien de modifier ces aspects pour résoudre le conflit. Deuxièmement, il nous semble intéressant d'étendre le mécanisme de caractères génériques proposés pour permettre d'utiliser ces caractères sur les noms des instances en plus de leurs utilisations sur les noms des messages.

Chapitre 3

Processus de Détection dans un Comportement Infini

3.1 Introduction et Présentation du Problème

Ce chapitre est la suite logique du précédent : nous continuons à présenter un mécanisme de détection de points de jonction (aussi appelé langage d'expression de coupe), mais cette fois dans des comportements infinis, c'est-à-dire dans des HMSCs contenant au moins un cycle. Puisque notre processus de tissage est statique, car il consiste à transformer un scénario en un autre scénario dans lequel les aspects ont été tissés, le problème de la détection dans des scénarios infinis est de trouver statiquement où se trouvent les points de jonction. L'ordre partiel induit par les MSCs et en particulier la nature hiérarchique[1] des HMSCs rendent nécessaire la considération du problème à un niveau sémantique, avec des techniques telles que le dépliage de cycle, etc.

Par la suite, nous appellerons *tissage syntaxique* d'un aspect comportemental $A = (P, Ad)$ dans un HMSC H, le tissage de A dans chaque bMSC étiquetant une transition de H.

Nous appellerons tissage *tissage sémantique* d'un aspect comportemental $A = (P, Ad)$ dans un HMSC H, le tissage de A dans tous les bMSCs générés par H, c'est-à-dire le tissage de A dans $\mathcal{L}(H) = \{\lambda(p) | p$ étant un chemin accepteur de $H\}$.

Pour mieux présenter le problème de la détection dans des scénarios infinis et les concepts que l'on introduira dans ce chapitre, nous nous servirons d'un exemple "fil conducteur". Considérons le HMSC H de la figure 3.1 où un client essaie de s'inscrire sur un serveur. Le serveur peut lui donner une réponse positive (le message *ok*), ou bien une réponse négative (le message *réessaie*) et dans ce dernier cas le client effectue une nouvelle tentative. Le client a la possibilité d'effectuer un nombre infini de tentatives puisque le bMSC *nvEssai* est dans une boucle.

Supposons maintenant que l'on souhaite tisser un comportement particulier dans ce système à chaque fois qu'un message *nouvelle tentative* précède directement un message *réessaie*. Pour spécifier ce genre de tissage, un aspect comportemental est très adapté.

[1] C'est-à-dire le fait qu'un HMSC compose des bMSCs

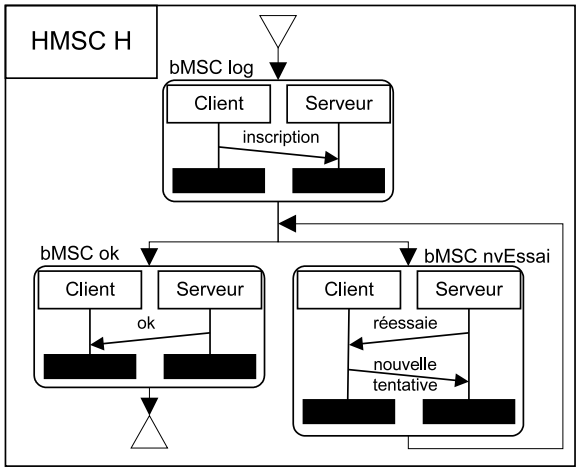


FIG. 3.1 – Un exemple d'HMSC

Premièrement, il permet de spécifier une expression de coupe avec un bMSC représentant le comportement à détecter. Deuxièmement, il permet de spécifier le comportement voulu avec un autre bMSC. La figure 3.2 représente l'aspect comportemental que l'on souhaite tisser dans le HMSC H de la figure 3.1.

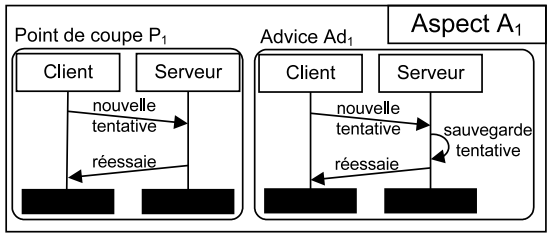


FIG. 3.2 – Un exemple d'aspect

Quand le tissage est effectué à un niveau syntaxique, puisque l'expression de coupe décrite par l'aspect A_1 n'apparaît pas explicitement dans le HMSC H, le HMSC reste inchangé (en effet, la suite de messages *nouvelle tentative-réessaie* n'apparaît ni dans le bMSC *log*, ni dans *ok* et ni dans *nvEssai*). Par contre, à un niveau sémantique, il est possible que des exécutions définies par le HMSC H contiennent des points de jonction correspondant à l'expression de coupe P_1. Par exemple, si l'on considère le comportement $log \bullet nvEssai \bullet nvEssai \bullet ok$ généré par le HMSC H, le point de jonction formé par la suite de messages *nouvelle tentative-réessaie* apparaît.

Dans ce chapitre, nous nous intéressons à la transformation du HMSC H vers le HMSC H' de la figure 3.3 qui a exactement la même sémantique que H (les deux HMSCs génèrent les mêmes comportements), mais où les points de jonction (représentés en pointillés rouge sur la figure 3.3) sont localisés dans un ensemble fini de chemins particuliers de H'. Autrement dit, de manière générale, nous présentons une transformation statique d'un HMSC H vers un HMSC H' équivalent, dans lequel la détection des points

de jonction correspondant à une expression de coupe donnée peut être effectuée en ne considérant qu'un ensemble fini de comportements partiels de H'.

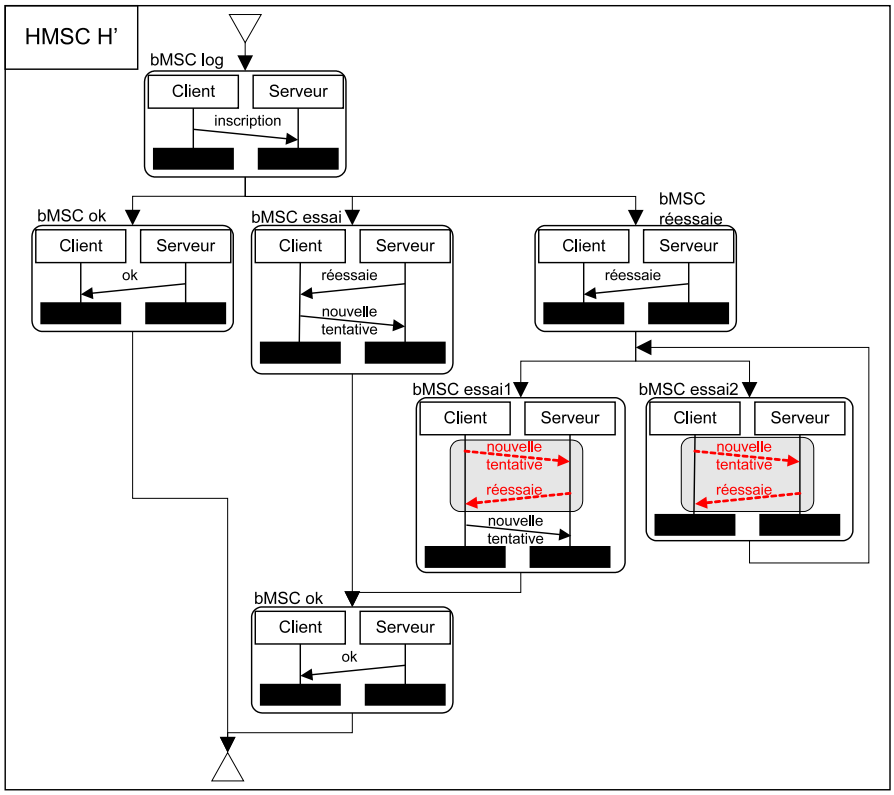


FIG. 3.3 – Résultat de la transformation

Ce chapitre est organisé comme suit : La section 3.2 précise la sémantique des points de jonction choisie et reformule le problème de détection. La section 3.3 introduit une nouvelle notion qui définit les "parties" commençant à être détectées par une expression de coupe à la fin d'un bMSC. La section 3.4 est la section centrale de ce chapitre. Elle présente plusieurs algorithmes permettant d'obtenir un ensemble fini de chemins dans lesquels une expression de coupe P est détectée. Les limitations de notre approche sont présentées dans la section 3.5. La section 3.6 conclut ce chapitre.

3.2 Choix de la sémantique des points de jonction et Reformulation de la problématique

Dans ce chapitre, nous présentons une solution pour détecter des points de jonction correspondants à une expression de coupe, uniquement pour des points de jonction définis comme des sous-bMSCs (nous ne présenterons pas de solution pour les notions de motif, motif sûr et motif clos). Ce choix est motivé par le fait que la définition de sous-bMSCs permet de définir de la manière la plus facile le problème de détection dans

des comportements infinis. Nous rappelons qu'un bMSC M' est un sous-bMSC d'un bMSC M s'il existe deux bMSCs X et Y tel que $M = X \bullet M' \bullet Y$ ($\bullet$ étant l'opérateur de composition séquentielle). Si nous ne considérons que la notion de sous-bMSC pour définir la notion de points de jonction, il n'est plus nécessaire d'utiliser des morphismes de bMSCs pour définir la notion de points de jonction. En effet, la définition de point de jonction peut être reformulée en :

Définition 3.1 (Point de Jonction) *Nous dirons qu'une expression de coupe P permet la détection d'un point de jonction J dans un bMSC M si et seulement si il existe deux bMSCs X_1 et X_2 tel que $M = X_1 \bullet J \bullet X_2$ et $J = P$. De plus, si X_1 est le plus petit bMSC permettant la décomposition de M, nous dirons que la détection est minimale.*

Par soucis de simplicité, nous écrirons directement $M = X_1 \bullet P \bullet X_2$ (P joue donc à la fois le rôle d'une expression de coupe et d'un point de jonction). Par léger abus, nous dirons également que "P est détecté dans M", au lieu de "P permet la détection d'un point de jonction J dans M. Enfin, nous noterons $P \triangleright M$ lorsque P est détecté dans M.

La définition de points de jonction successifs peut également être reformulée :

Définition 3.2 (Point de Jonction successifs) *Une expression de coupe P est détectée k fois dans un bMSC M (noté par $P \triangleright^k M$) s'il existe $k+1$ bMSCs $\{X_i\}_{i \in \{1..k+1\}}$ tel que :*

- $M = X_1 \bullet P \bullet X_2 \bullet ... \bullet X_k \bullet P \bullet X_{k+1}$;
- $\forall i, X_i \bullet P \bullet X_{i+1}$ *la détection est minimale. Cette propriété garantit l'unicité de la détection : les points de jonction successifs sont détectés en séquence dès qu'ils apparaissent dans le bMSC.*

Pour étendre la notion de détection (et donc de tissage) aux HMSCs, nous devons résoudre le problème suivant : pour un chemin $p = t_1 \dots t_n$ d'un HMSC H, quand $P \triangleright^q \lambda(p)$, toutes les occurrences de P ne sont pas nécessairement détectées dans une seule transition. En effet, une partie de P peut être détectée dans le bMSC étiquetant la transition $\lambda(t_i)$, et complétée plus tard dans un bMSC étiquetant une autre transition $\lambda(t_{i+k})$. De plus, nous devons identifier toutes les parties du HMSC de base où un point de jonction peut apparaître sans pour autant déplier l'ensemble potentiellement infini des comportements que le HMSC peut générer. Considérons un HMSC H et un aspect comportemental $A = (P, Ad)$. Le but de la détection est de transformer un HMSC de base H en un HMSC équivalent H' (c'est-à-dire $\mathcal{L}(H) = \mathcal{L}(H')$) de telle manière que l'expression de coupe P est seulement détectée dans un nombre fini de chemins particuliers de H'.

3.3 Détection Potentielle

Dans cette section, nous introduisons une nouvelle notion qui nous sera utile pour détecter des points de jonction dans des comportements infinis. Pour une expression de coupe P donnée et un bMSC M lié à une étiquette d'une transition t d'un HMSC H, la

notion, appelée *détection potentielle*, indique les parties de M qui "commencent" à être reconnues par P. Grâce à cette indication, nous saurons s'il est nécessaire de continuer la recherche de P dans les bMSCs étiquetant les transitions qui suivent t.

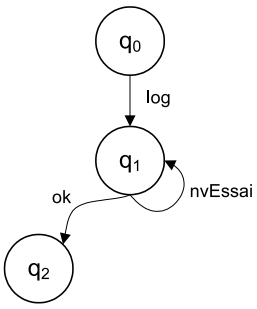


FIG. 3.4 – Représentation en automate du HMSC H de la figure 3.1

Mais avant de donner une définition formelle de la *détection potentielle*, intéressons-nous à un exemple. Considérons le HMSC H de la Figure 3.1, et sa représentation en automate de la Figure 3.4. Considérons également l'expression de coupe $P1$ de la Figure 3.2. Nous devons détecter un message "nouvelle tentative" suivi par un message "réessaie" dans toutes les exécutions de H. Pour explorer les chemins de H, nous débutons par le noeud initial de H, et nous regardons si $P1$ est détecté dans la transition étiquetée par le bMSC *log*. Puisque $P1$ n'est pas détecté dans *log* et puisque $P1$ ne commence pas à être détecté dans *log*, nous pouvons continuer notre exploration à partir du noeud q_1, et regarder les transitions suivantes, c'est-à-dire les transitions étiquetées par le bMSC *ok* et le bMSC *nvEssai*. $P1$ n'est pas détecté dans *ok* et puisque q_2 est un noeud final, nous savons que $P1$ n'est jamais détecté après le noeud q_2. $P1$ n'est également pas détecté entièrement dans *nvEssai*, mais il existe un début de détection, car le message "nouvelle tentative" est détecté à la fin du bMSC *nvEssai*. Donc, l'expression de coupe pourrait être détectée dans un bMSC plus long. Comme il y a deux transitions $t_1 = (q_1, ok, q_2)$ et $t_2 = (q_1, nvEssai, q_1)$ dans H, selon la sémantique des HMSCs, nous pouvons rechercher une détection éventuelle de $P1$ dans les bMSCs $nvEssai \bullet ok$ et $nvEssai \bullet nvEssai$. $P1$ n'est pas détecté dans $nvEssai \bullet ok$, mais par contre, $P1$ est détecté dans $nvEssai \bullet nvEssai$, et un nouveau début de détection apparaît dans le bMSC $nvEssai \bullet nvEssai$.

La définition de points de jonction successifs (définition 3.2) indique le nombre de fois qu'une expression de coupe P est entièrement détectée dans un bMSC M. En plus de cette définition, comme nous le montre l'exemple décrit précédemment, il est également nécessaire d'obtenir des informations sur la notion de début de détection d'une expression de coupe dans un bMSC M, cette détection pouvant éventuellement être complète dans une extension $M \bullet M'$. Pour cela, nous introduisons une nouvelle notion appelée *détection potentielle*. Comme première approximation, pour un bMSC M et une expression de coupe P, une détection potentielle $PM_{P,M}$ est la plus grande concaténation de parties de M dans lesquelles P est partiellement détectée et qui peut être éventuellement complétée pour que P soit complètement détecté dans M.

La Figure 3.5 montre un exemple où l'expression de coupe $P2$ est détectée une fois dans le bMSC M. M peut être écrit $X_1 \bullet P2 \bullet X_2$, mais $P2$ commence à être détecté à la "fin" de M. En effet, si nous ajoutons le bMSC N à M, $P2$ est détecté trois fois dans $M \bullet N$, et à chaque fois les nouvelles détections impliquent des événements de M. La partie de M qui participe à ces deux détections supplémentaires forme une détection potentielle, et elle est représentée par la partie notée $PM_{P2,M}$ dans la Figure 3.5.

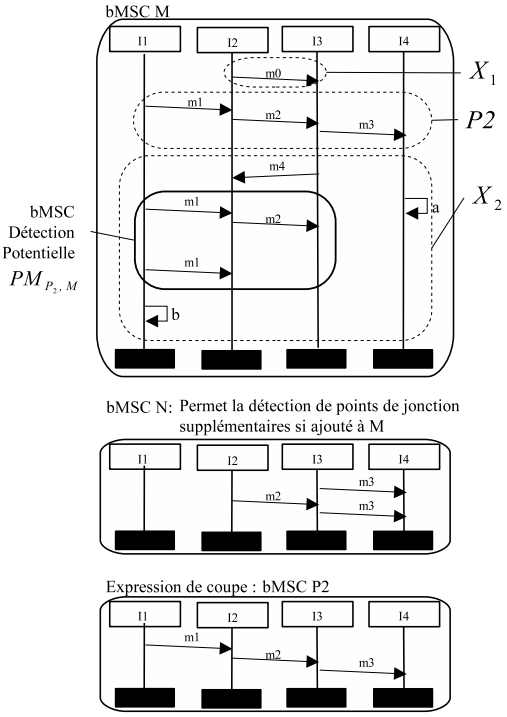


FIG. 3.5 – Un exemple de détection potentielle

La notion de détection potentielle peut être formellement définie par :

Définition 3.3 (Détection Potentielle) *Soient P et M deux bMSCs tels que P soit détecté k fois dans M. Nous pouvons alors écrire $M = X_1 \bullet P \bullet X_2 \bullet ... \bullet X_k \bullet P \bullet X_{k+1}$. Nous appellerons une* détection potentielle *d'une expression de coupe P dans un bMSC M, notée $PM_{P,M}$, le bMSC $PM_{P,M} = P_{1,1} \bullet P_{2,1} \bullet ... \bullet P_{q,1}$ contenu dans le bMSC X_{k+1} tel que :*

- *$P_{1,1}$ est le plus grand bMSC détecté dans X_{k+1} tel que $\exists P_{1,2}, P = P_{1,1} \bullet P_{1,2}$ et $P \triangleright X_{k+1} \bullet P_{1,2}$;*
- *$\forall i \in \{2...q\}$, $P_{i,1}$ est le plus grand bMSC détecté dans X_{k+1} après $P_{i-1,1}$ tel que $\exists P_{i,2}, P = P_{i,1} \bullet P_{i,2}$ et $P \triangleright^i X_{k+1} \bullet P_{1,2} \bullet P_{2,2} \bullet ... \bullet P_{i,2}$.*

Les bMSCs $P_{i,1}$ sont appelés les parties potentielles de la détection potentielle $PM_{P,M}$.

La figure 3.6 détaille les parties potentielles $P_{1,1}$ et $P_{2,1}$ liées au bMSC M et à l'expression de coupe $P2$.

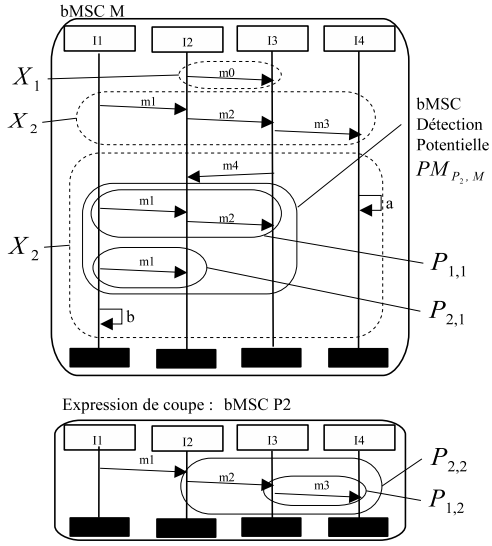


FIG. 3.6 – Les parties potentielles

Une détection potentielle indique que la détection débutée doit être étudiée sur de plus longues exécutions, c'est-à-dire que les chemins considérés doivent être étendus pour obtenir de plus grands bMSCs dans lesquels P est complètement détecté.

3.4 Obtention des chemins dans lesquels l'expression de coupe est détectée

Pour une expression de coupe P et un HMSC de base H, le but est de transformer H en un HMSC équivalent H' tel que la détection est à effectuer uniquement dans un nombre fini de chemins de H'. Cette transformation de H en H' et l'obtention de tous les chemins dans lesquels l'expression de coupe est détectée, sont effectuées grâce à plusieurs transformations simples. Avant de détailler ces transformations, nous commençons par en donner une brève description en quatre points :

1. La première étape (implémentée par l'algorithme 4) consiste à déplier le HMSC de base H pour exhiber toutes les détections potentielles de l'expression de coupe et calculer un HMSC équivalent H_+ tel que chaque noeud de H_+ est une paire (q, C) où q est un noeud de H, et C est une détection potentielle de l'expression de coupe P pour un chemin initial qui finit sur le noeud q. Par la suite, nous appellerons cette détection potentielle le *contexte* de q. Grâce à cette première étape, pour chaque noeud q avec un contexte, le contexte indique s'il existe des débuts de détection et donc s'il est nécessaire de continuer l'étude de ces débuts de détection sur des chemins plus longs.
2. Cependant, comme nous allons le voir en sous-section 3.4.2, quand P commence

à être détecté dans un chemin qui conduit à un contexte C, C n'est pas toujours entièrement utilisé lors d'une détection. Pour cette raison, dans une deuxième étape, nous devons calculer les parties du contexte (ou de la détection potentielle) qui sont réellement utilisées lors de la détection de P. Ces parties seront appelées les *détections futures* (ou simplement futurs) contenues dans C. Cette seconde étape est effectuée par l'algorithme 5 qui transforme un HMSC H_+ en un HMSC équivalent $H_\times$, tel que chaque noeud de $H_\times$ est un triplet (q, C, F), où (q, C) est un noeud de H_+ et F est une détection future contenue dans C. De cette manière, pour chaque noeud q, la détection future indique si les débuts de détections présents dans le contexte associé à q, se termineront réellement dans des chemins plus longs.

Notons cependant que ces deux transformations ont quelques limites qui seront décrites en section 3.5

3. L'algorithme 6 permet la résolution de problèmes liés aux cycles dans les HMSCs (sous-section 3.4.3 et 3.4.4) : dès qu'un HMSC $H_\times$ contient au moins un cycle c tel que tous les noeuds de c ont une détection future non vide, la phase de détection ne peut se terminer sans effectuer de permutations utiles de bMSCs.

4. Finalement, quand l'algorithme 6 termine correctement, l'algorithme 7 (sous-section 3.4.5) construit l'ensemble $\mathcal{P}_m$ des chemins acycliques minimaux dans lesquels l'expression de coupe est détectée entièrement.

La figure 3.7 résume le processus de transformation d'un HMSC de base H en un HMSC H' dans lequel la détection d'une expression de coupe P a lieu uniquement dans un nombre fini de chemins de H'.

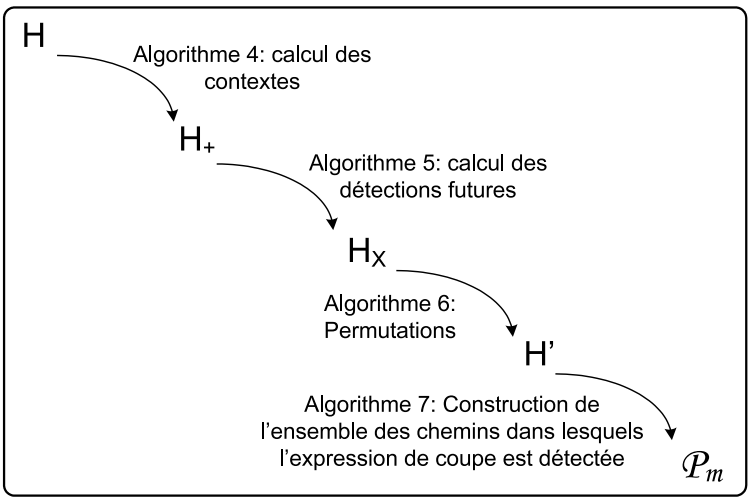


FIG. 3.7 – Processus de transformation de H permettant d'obtenir un ensemble fini de chemins dans lesquel une expression de coupe est détectée

3.4.1 Calcul des Détections Potentielles

La première transformation, implémentée par l'algorithme 4, consiste à déplier un HMSC de base H. Cet algorithme transforme H en un HMSC équivalent H_+ tel que chaque noeud de H_+ est une paire (q, C) où q est un noeud de H, et C est une détection potentielle (aussi appelée *contexte*) de l'expression de coupe P pour un chemin initial qui fini sur le noeud q. Par exemple, pour une expression de coupe P et un chemin initial $p = (q_0, M1, q_1).(q_1, M2, q_2).(q_2, M3, q_3)$, nous construirons les noeuds (q_0, M_ϵ), $(q_1, PM_{P,M1})$, $(q_2, PM_{P,M1 \bullet M2})$, et $(q_3, PM_{P,M1 \bullet M2 \bullet M3})$. Ces nouveaux noeuds peuvent seulement être calculés les uns après les autres. La première paire $(noeud, contexte)$ créée est (q_0, M_ϵ) où M_ϵ est le bMSC vide (puisqu'il n'y a pas de chemin avant q_0, le contexte est forcément nul). Ensuite, nous pouvons calculer $PM_{P,M1}$ et construire le noeud $(q_1, PM_{P,M1})$. Pour calculer le contexte $PM_{P,M1 \bullet M2}$ du noeud q_2, nous pouvons utiliser la propriété suivante :

Propriété 3.1 *Soient deux bMSCs X et Y, et une expression de coupe P. Le contexte de $X \bullet Y$ est égal au contexte de $PM_{P,X} \bullet Y$, autrement dit $PM_{P,X \bullet Y} = PM_{P,PM_{P,X} \bullet Y}$*[2].

Donc, pour calculer le contexte $PM_{P,M1 \bullet M2}$ du noeud q_2, il n'est pas nécessaire de prendre en compte le bMSC $M1$ en entier : la prise en compte du contexte $PM_{P,M1}$ du noeud $q1$ créé précédemment est suffisante. Finalement, pour calculer $PM_{P,M1 \bullet M2 \bullet M3}$, conformément à la propriété 3.1, nous pouvons utiliser un contexte calculé précédemment relatif au noeud q_2, c'est-à-dire, $PM_{P,M1 \bullet M2}$, et nous calculons simplement $PM_{P,PM_{P,M1 \bullet M2} \bullet M3}$. De cette manière, à partir du noeud initial, chaque contexte peut être calculé de manière inductive : pour calculer les contextes relatifs au noeud q, il n'est pas nécessaire d'étudier tous les chemins initiaux qui finissent sur q, mais seulement les contextes déjà calculés qui sont attachés aux prédécesseurs immédiats du noeud q.

L'algorithme 4 utilise l'approche qui vient juste d'être décrite pour calculer les contextes d'un HMSC H. Puisque cet algorithme est juste un dépliage du HMSC initial, il est évident que les ensembles des exécutions générées par H et H_+ sont équivalents. Il faut cependant noter que cet algorithme se termine si et seulement si le nombre de détections potentielles (ou de contextes) pour chaque noeud de H est fini ($\mathcal{N}$ étant fini par définition), ce qui n'est pas toujours le cas. Ce problème sera développé dans la section 3.5. Nous montrerons que si le bMSC expression de coupe est un bMSC connexe, le nombre de contextes pour chaque noeud de H est forcément fini.

Pour l'expression de coupe $P1$ de la figure 3.2, l'algorithme 4 transforme le HMSC H de la figure 3.8 (qui est la représentation en automate du HMSC H de la figure 3.1) vers le HMSC H_+ de la même figure. Dans la figure 3.8, M_ϵ représente le bMSC vide et PM un bMSC avec uniquement un message "nouvelle tentative" de l'instance *client* à l'instance *serveur*. Le contexte associé à q_1 pour le chemin $p = (q_0, log, q_1)$ est M_ϵ, puisque $P1$ n'est détecté ni entièrement ni partiellement dans log. Pour chaque chemin terminant avec la transition $(q_1, nvEssai, q_1)$, le contexte associé à q_1 est PM puisque le message "nouvelle tentative" est détecté par $P1$.

[2] La démonstration de cette propriété est triviale, puisque les événements de X pouvant participer à une détection future, appartiennent forcément au contexte associé à X.

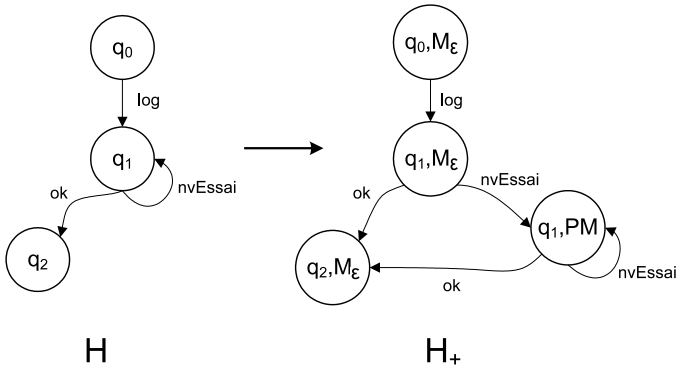


FIG. 3.8 – Un exemple de transformation

Algorithme 4 Contexte(P,H)

entrées : expression de coupe P, HMSC $H = (\mathcal{N}, T, q_0, \mathcal{Q}_{end}, \mathcal{M})$
sortie : HMSC $H_+ = (\mathcal{N}_+, T_+, q_{+0}, \mathcal{Q}_{end_+}, \mathcal{M}_+)$

1: $\mathcal{M}_+ = \mathcal{M}$, $q_{+0} = (q_0, M_\epsilon)$
2: $NoeudCourrant = \{q_{+0}\}, NoeudFutur = \emptyset$,
3: $\mathcal{N}_+ = \{q_{+0}\}, T_+ = \emptyset$
4: **tant que** $NoeudCourrant \neq \emptyset$ **faire**
5: **pour tout** $q_+ = (q, M_p) \in NoeudCourrant$ et $t = (q, M, q') \in T$ **faire**
6: $q'_+ = (q', PM_{P,M_p \bullet M})$
7: **si** $q'_+ \notin \mathcal{N}_+$ **alors**
8: $NoeudFutur = NoeudFutur \cup q'_+$
9: $\mathcal{N}_+ = \mathcal{N}_+ \cup q'_+$,
10: **fin si**
11: $t_+ = (q_+, M, q'_+)$
12: **si** $t_+ \notin T_+$ **alors**
13: $T_+ = T_+ \cup t_+$
14: **fin si**
15: **fin pour**
16: $NoeudCourrant = NoeudFutur$,$NoeudFutur = \emptyset$
17: **fin tant que**
18: $\mathcal{Q}_{end_+} = \{(n, M) \in \mathcal{N}+ \mid n \in \mathcal{Q}_{end}\}$
19: retour $H_+ = (\mathcal{N}_+, T_+, q_{+0}, \mathcal{Q}_{end_+}, \mathcal{M}_+)$

3.4.2 Détections Futures

Pour une expression de coupe P et deux bMSCs M et M', le contexte $PM_{P,M}$ n'est pas toujours entièrement utilisé quand P est détecté dans $M \bullet M'$. Considérons par exemple le bMSC M, l'expression de coupe $P2$ et le bMSC M' de la figure 3.9. Le bMSC $M \bullet M'$ est représenté dans la figure 3.9 et nous pouvons constater que la

partie du contexte $C = PM_{P,M}$ qui est réellement utilisée lors de la détection de P dans $M \bullet M'$ est uniquement formée du message $m1$. Cette partie du contexte de C sera appelée *détection future* (ou simplement futur) dans le contexte C quand M' suit, et elle sera notée $FM_{P,C,M'}$.

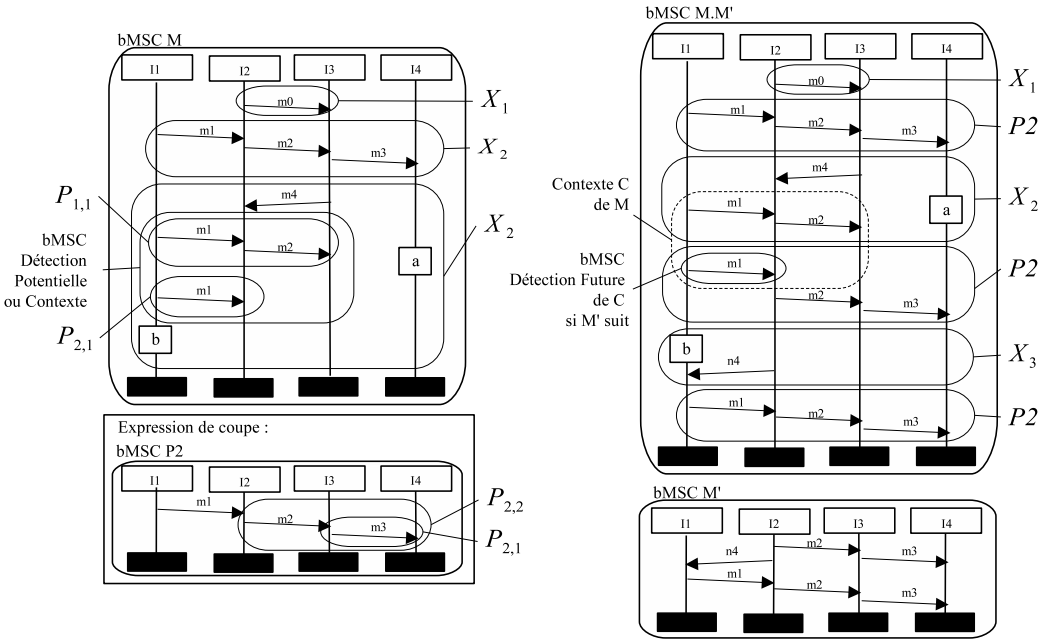


FIG. 3.9 – Un exemple de détection future

De manière similaire, pour un noeud quelconque $q_+ = (q, C)$ d'un HMSC H, la partie de C qui sera effectivement utilisée pour formée un point de jonction (ou utilisée lors de la détection de l'expression de coupe) dépends des chemins qui peuvent partir de q_+. Si $q_+ \in \mathcal{Q}_{end_+}$, il est clair que la détection future associée au noeud (q, C) est M_ϵ, puisque aucune extension d'un chemin finissant sur q_+ est possible. Si $q_+ \notin \mathcal{Q}_{end_+}$, alors la partie de C dans laquelle une expression de coupe est détectée dépends des transitions qui suivent. Même si le nombre de chemins partant d'un noeud q_+ et arrivant sur un noeud final peut être infini, l'ensemble des détections futures peut être défini de manière inductive comme suit :

Définition 3.4 (Détection Future) *Soient un HMSC H_+ (avec contexte), une expression de coupe P, et un noeud $q_+ = (q, C)$ de H_+. Une* détection future *associée au noeud q_+ est un sous-ensemble $F(q_+)$ de C défini comme suit :*

1. $F(q_+) = M_\epsilon$ *si* $q_+ \in \mathcal{Q}_{end_+}$,
2. $F(q_+) = FM_{P,C,M} \cup (C \cap F')$ *si* $q_+ \notin \mathcal{Q}_{end_+}$, *et s'il existe un noeud* $q'_+ = (q', C')$ *et une transition* $t = (q_+, M, q'_+) \in T_+$ *tel que* F' *est une détection future de* q'_+.

Le second point de la définition mérite d'être précisé. Les éléments qui apparaissent dans $FM_{P,C,M}$ forme la partie du contexte C qui est réellement détectée si le bMSC M suit (partie formant réellement un point de jonction). Notons cependant que d'autres événements de C peuvent être détecté avec un bMSC plus grand, mais ils apparaissent alors nécessairement dans la détection future F' associée à q'_+. Donc ces autres événements sont contenus dans $C \cap F'$.

L'algorithme 5 permettant de calculer tous les détections futures possibles pour un HMSC avec contexte découle immédiatement de la définition 3.4. Cet algorithme transforme un HMSC H_+ en un HMSC équivalent $H_\times$ tel que chaque noeud de $H_\times$ est un triplet (q, C, FM) où (q, C) est un noeud de H_+, et FM est une détection future pour le contexte C. Comme l'algorithme 4, la transformation implémentée par l'algorithme 5 est un dépliage de H_+. Elle produit donc un HMSC équivalent (l'ensemble des comportements générés sont préservés). La principale différence est que l'algorithme 5 part des noeuds finaux alors que l'algorithme 4 part du noeud initial.

Nous allons illustrer l'utilisation de l'algorithme 5 sur l'exemple de la figure 3.10. Considérons le HMSC H_0 et l'expression de coupe P_0 de cette figure. La figure 3.11 présente H_0 comme un automate étiqueté par des noms de bMSCs, et les HMSCs H_{0_+} et $H_{0_\times}$ obtenus en appliquant successivement les algorithmes 4 et 5. Dans H_{0_+}, $C1$ est le contexte du noeud q_1. $C1 = X$, et après chaque transition (q_1, Y, q_1), $C1$ reste inchangé. Dans $H_{0_\times}$, le noeud $(q1, C1)$ est dupliqué avec deux détections futures différentes, puisque la partie du bMSC X qui sera détecté par P_0 n'est pas la même lorsque Z suit immédiatement X et lorsqu'au moins une itération de Y est insérée entre X et Z. La figure 3.12 montre un autre exemple d'un HMSC $H_\times$ obtenu à partir du HMSC H_+ de la figure 3.8.

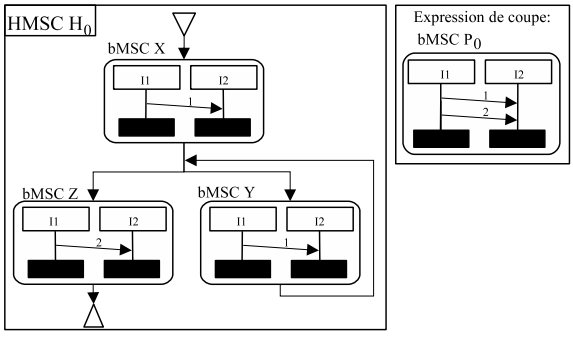


FIG. 3.10 – Un HMSC

Notons que si l'algorithme 4 termine, alors l'algorithme 5 termine également. En effet, si le nombre de détections potentielles est borné pour un HMSC H et une expression de coupe P donnés (condition pour que l'algorithme 4 converge), alors le nombre de détection future est également borné, car une détection future est toujours un sous-ensemble d'une détection potentielle.

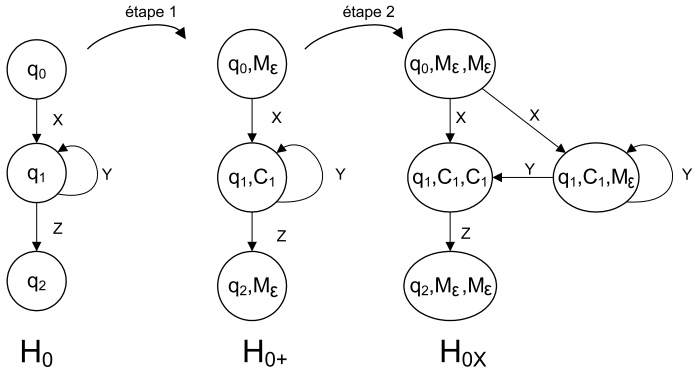


FIG. 3.11 – Transformation de H vers $H_\times$

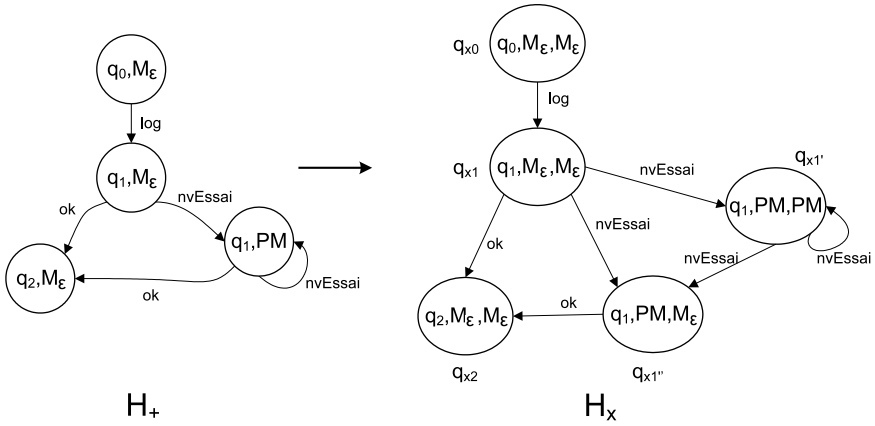


FIG. 3.12 – HMSC $H_\times$ obtenu à partir du HMSC H_+ de la figure 3.8

Algorithme 5 DétectionFuture(P,H_+)

entrées : expression de coupe P, HMSC $H_+ = (\mathcal{N}_+, T_+, q_{+0}, \mathcal{Q}_{end_+}, \mathcal{M}_+)$
sortie : HMSC $H_\times = (\mathcal{N}_\times, T_\times, q_{\times 0}, \mathcal{Q}_{end_\times}, \mathcal{M}_\times)$

1: $\mathcal{M}_\times = \mathcal{M}_+$, $\mathcal{Q}_{end_\times} = \{(n, M, M_\epsilon) | (n, M) \in\in \mathcal{Q}_{end_+}\}$
2: $NoeudCourrant = \mathcal{Q}_{end_\times}, NoeudFutur = \emptyset$,
3: $\mathcal{N}_\times = NoeudCourrant, T_\times = \emptyset$
4: **tant que** $NoeudCourrant \neq \emptyset$ **faire**
5: **pour tout** $q_\times = (q, C, F) \in NoeudCourrant$ et $t_+ = ((q', C'), M, (q, C)) \in T_+$ **faire**
6: $F' = FM_{P,C',M} \cup (C' \cap F)$
7: $q'_\times = (q', C', F')$
8: **si** $q'_\times \notin \mathcal{N}_\times$ **alors**
9: $NoeudFutur = NoeudFutur \cup q'_\times$
10: $\mathcal{N}_\times = \mathcal{N}_\times \cup q'_\times$,
11: **fin si**
12: $t_\times = (q'_\times, M, q_\times)$
13: **si** $t_\times \notin T_\times$ **alors**
14: $T_\times = T_\times \cup t_\times$
15: **fin si**
16: **fin pour**
17: $NoeudCourrant = NoeudFutur$,$NoeudFutur = \emptyset$
18: **fin tant que**
19: $q_{\times 0} = (q_0, M_\epsilon, M_\epsilon)$,
20: retour $H_\times$

3.4.3 Cycles Problématiques

Pour une expression de coupe P et un HMSC H, nous avons construit un HMSC $H_\times$ dont les noeuds contiennent un contexte et un futur. Comme nous voulons isoler les points de jonction dans un nombre fini de chemins, l'étape suivante consiste à réécrire $H_\times$ en un HMSC équivalent où le nombre de chemins partant d'un noeud avec un futur vide, arrivant sur noeuds avec un futur vide et ne passant jamais par d'autres noeuds avec un futur vide est fini. L'algorithme 7 construit l'ensemble $\mathcal{P}_m$ de ces chemins et les définit de manière plus formelle. Cet ensemble $\mathcal{P}_m$ assure que P est détecté entièrement dans le bMSC $\lambda(p)$ associé au chemin p de $\mathcal{P}_m$ un certain nombre de fois, et qu'aucune détection future ne débute à la fin de ce chemin (la détection future associée à ce chemin est nulle).

Cependant, l'algorithme 7 ne peut pas toujours être directement appliqué à un HMSC $H_\times$ avec contexte et détection future. Si $H_\times$ contient au moins un cycle c tel que tous les noeuds de c ont un futur différent du bMSC vide, alors l'algorithme diverge. Considérons par exemple le HMSC de la figure 3.12 : il y a un nombre infini de chemins partant et arrivant sur une détection future vide : $nvEssai \bullet nvEssai$, $nvEssai \bullet nvEssai \bullet nvEssai$, etc...

Pour cette raison, avant d'utiliser l'algorithme 7, nous devons modifier ou casser les cycles de $H_\times$ qui bouclent infiniment sur des noeuds avec des futurs vides (sans changer le comportement de $H_\times$). Notons par $\mathcal{C}_{H_\times}$ l'ensemble des cycles élémentaires de $H_\times$, et $\mathcal{UC}_{H_\times}$ l'ensemble des cycles élémentaires problématiques. Formellement, nous avons $\mathcal{UC}_{H_\times} = \{c = t_1.t_2...t_k \in \mathcal{C}_{H_+} | \forall i \in \{1...k\}, FM(\lambda_-(t_i)) \neq M_\epsilon\}$. Pour qu'un cycle ne soit plus problématique, il est possible que la résolution consiste seulement à découper un bMSC étiquetant une transition en de plus petits bMSCs entre lesquels un futur vide peut apparaître. Par exemple, dans la figure 3.12, la transition $t = (q_{\times_{1'}}, nvEssai, q_{\times_{1'}})$ peut être découpée en deux transitions $t' = (q_{\times_{1'}}, nvEssai1, q_{new})$ et $t'' = (q_{new}, nvEssai2, q_{\times_{1'}})$ tel que $nvEssai1$ est un bMSC avec uniquement un message "réessaie" et $nvEssai2$ est un bMSC avec uniquement message "nouvelle tentative". Après cette transformation, on peut facilement noter que le futur du noeud q_{new} est le bMSC vide M_ϵ. La figure 3.13 décrit cette "découpe" simple de bMSCs

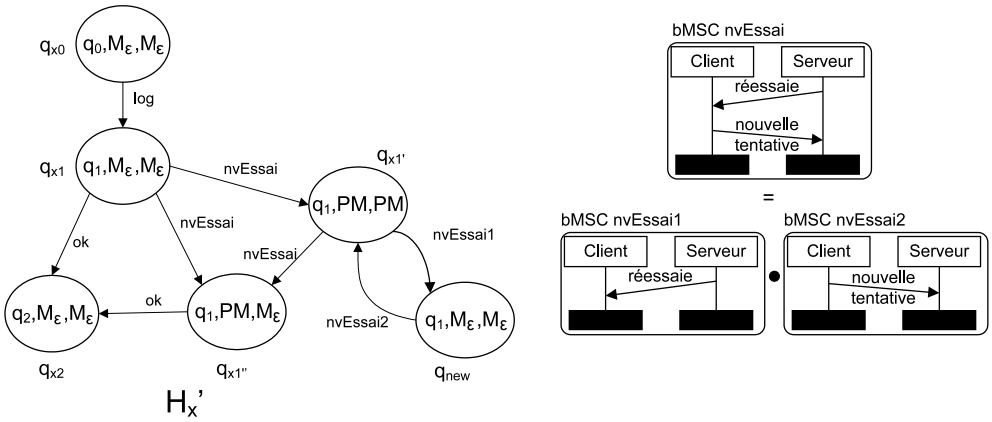


FIG. 3.13 – Transformation simple faisant apparaître un futur vide

Cependant, les transformations des cycles problématiques ne sont pas toujours si faciles. Considérons le HMSC H_3 de la figure 3.14. L'expression de coupe $P3$ est détectée dans un nombre infini de bMSCs $(X1X2, X1Y1X2, X1Y1Y1X2, ...)$, mais pour chaque point de jonction J présent dans ces bMSCs, les événements de $Y1$ relatifs au cycle ne sont jamais utilisés pour former J. L'algorithme 7 ne peut pas être appliqué à un HMSC contenant ce genre de cycle, car il y a un nombre infini de chemins partant et arrivant sur des noeuds à futur vide. Puisque les bMSCs $X1$ et $Y1$ sont indépendants $(X1 \bullet Y1 = Y1 \bullet X1)$, une solution possible est de réécrire le HMSC de la figure 3.14 en un HMSC équivalent de telle manière que le cycle sur $Y1$ soit placé avant le bMSC $X1$. Nous obtenons ainsi le HMSC $H3'$ de la figure 3.14. Notons que dans ce nouveau HMSC, les détections de $P3$ n'impliquent plus un nombre infini d'occurrences de $Y1$. Nous allons maintenant formaliser cette transformation, et l'appliquer de manière systématique à la résolution du problème posé par les cycles.

Définition 3.5 (Générateur Problématique) *Soient $H_\times$ un HMSC avec contextes et futurs, et P une expression de coupe. Soit $\mathcal{UC}_{H_\times}$ l'ensemble des cycles problématiques*

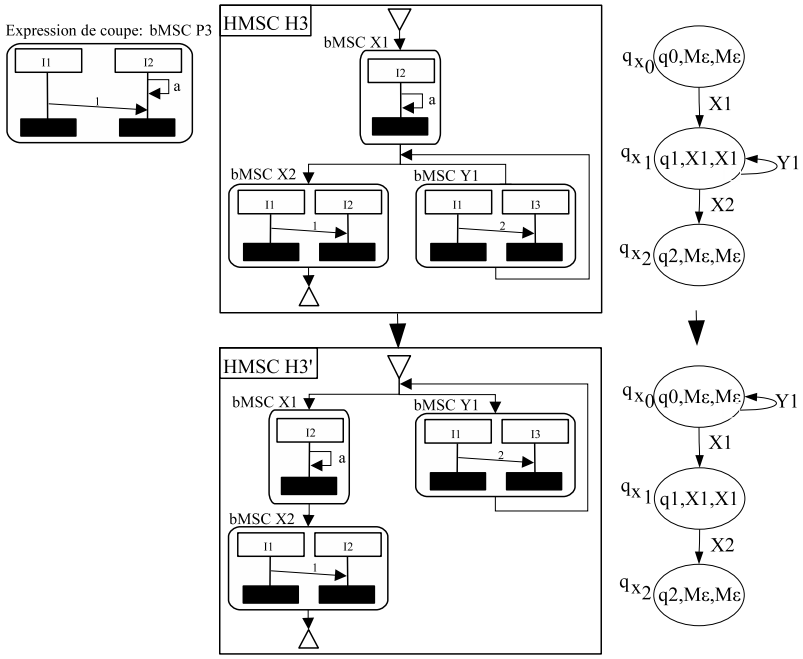


FIG. 3.14 – Un problème de cycle résolu avec une permutation

de $H_\times$. *Un* générateur problématique *de* $H_\times$ *et* P *est un chemin acyclique* $p = t_1 \dots t_k$ *de* $H_\times$ *qui :*

- *part d'un noeud avec un futur vide, c'est-à-dire* $F(\lambda_-(t_1)) = M_\epsilon$
- *arrive sur un noeud avec un futur vide, c'est-à-dire* $F(\lambda_+(t_k)) = M_\epsilon$
- *ne passe jamais sur un noeud avec un futur vide autre que* $\lambda_-(t_1)$ *et* $\lambda_+(t_k)$, *c'est-à-dire* $\forall i \in 2..k$, $F(\lambda_-(t_i)) \neq M_\epsilon$
- *traverse un cycle problématique de* $H_\times$, *c'est-à-dire* $\exists j \in 2..k, c = t'_1 \dots t'_q \in \mathcal{UC}_{H_\times}$ *tel que* $\lambda_-(t_j) \in \bigcup_{i \in 1..q} \lambda_-(t'_i)$
- P *est détecté dans* $\lambda(p)$

Soit $\mathcal{P}_u(H_\times)$ l'ensemble des générateurs problématiques de $H_\times$. L'idée principale de l'algorithme qui va suivre est de vérifier s'il existe un HMSC équivalent, à des permutations de bMSCs indépendants près, qui ne contient pas de générateurs problématiques.

3.4.4 Transformer les générateurs problématiques grâce à des permutations

Jusqu'ici, nous avons identifié un problème lié à des chemins acycliques qui traversent des cycles dans lesquels une expression de coupe est détectée et dans lesquels tous les noeuds ont un futur non vide. A cause de ce problème et lorsque des générateurs problématiques existent, la détection d'une expression de coupe ne consiste pas uniquement à déplier un HMSC initial. L'objectif principal consiste alors à trouver s'il existe une transformation des chemins problématiques en une séquence de transitions équivalentes

(en termes de comportements générés) tel que les cycles passent systématiquement à travers un noeud avec futur vide.

La transformation des chemins problématiques est mieux décrite en utilisant des expressions régulières. Nous rappelons tout d'abord qu'un bMSC B est appelé un *atome* s'il n'existe pas deux bMSCs $N \neq M_\epsilon$ et $M \neq M_\epsilon$ tel que $B = M \bullet N$.

Définition 3.6 (Expression Régulière) *Une expression régulière sur un alphabet d'atomes Σ_{At} est une expression de la forme $E ::= \sigma.E \mid (E)^* \mid E1 + E2 \mid M_\epsilon$, où $\sigma \in \Sigma_{At}$ est un bMSC atomique, $(E)^*$ représente l'itération, et $E1 + E2$ l'alternative.*

Les expressions régulières et les automates finis ont le même pouvoir d'expression. A partir d'une expression régulière, nous pouvons calculer un automate générant le même langage (en utilisant l'algorithme de Brzozowski que l'on trouvera par exemple dans [HU79]), et à partir d'un automate, nous pouvons calculer une expression équivalente (en utilisant un algorithme de réduction). De cette manière, nous pouvons manipuler indifféremment les HMSCs ou les expressions régulières de bMSCs, et passer facilement d'un modèle à un autre. Pour une expression régulière donnée E, nous nommerons H_E le HMSC générant le même ensemble de scénarios, et pour un HMSC H donné, nous nommerons E_H l'expression régulière équivalente définie sur le même ensemble de bMSCs. Le chapitre 2 du livre de Hopcroft et Ullman [HU79] donne des détails supplémentaires sur les relations entre automates et expressions régulières.

De manière similaire à ce qui a été proposé pour les HMSCs, nous pouvons associer un contexte à une expression $E = X_1 \dots X_k$, en calculant la détection potentielle d'une expression de coupe P dans $X_1 \bullet \dots \bullet X_k$. De même, pour toute sous-expression $E' = X_1 \dots X_i$ de E, nous pouvons aussi calculer une détection future $Futur(E')$ avec $Futur(E') = FM_{P,E',X_{i+1}\bullet\dots X_k}$.

Cette section fournit une solution pour les cas où les cycles problématiques sont *disjoints*, c'est-à-dire quand $\forall c = t_1 \dots t_k, c' = t'_1 \dots t'_p \in \mathcal{UC}_{H_\times}$, $\bigcup_{i\in 1..k} \lambda_-(t_i) \cap \bigcup_{i\in 1..p} \lambda_-(t'_i) = \emptyset$. Avec cette restriction, si $p = t_1.t_2.\dots.t_q$ est un générateur problématique lié à un ensemble disjoint de cycles problématiques $C = \{c_1, c_2, \dots c_k\}$, nous pouvons associer à p une expression :

$$E_p = \quad X_{11} \dots X_{1m_1}(Y_{11} \dots Y_{1n_1})^*.X_{21} \dots X_{2m_2}.(Y_{21} \dots Y_{2n_2})^* \dots (Y_{k1} \quad \dots Y_{kn_k})^*.X_{g1} \dots X_{gm_q}$$

tel que :

- X_{ij} et Y_{ij} sont des atomes,
- E_p génère le même ensemble de comportements que $\lambda(t_1) \bullet \dots \bullet \lambda(t_{i_1}) \bullet \lambda(c_1)^* \bullet \lambda(t_{i_1+1}) \bullet \dots \bullet \lambda(t_{i_k}) \bullet \lambda(c_k)^* \bullet \lambda(t_{i_k+1}) \bullet \dots \bullet \lambda(t_q)$,
- pour tout $i_j, j \in 1..k, \lambda_+(t_{i_j}) = \lambda_-(c_j)$.

Notons que comme les cycles problématiques sont disjoints, cette expression n'est jamais de la forme $E1 + E2$ (forme d'expression qui n'est pas traité par l'algorithme 6).

Considérons par exemple le HMSC $H3$ et l'expression de coupe $P3$ de la figure 3.14. Le chemin $p = (q_{\times_0}, X_1, q_{\times_1})(q_{\times_1}, X_2, q_{\times_2})$ est un générateur problématique puisque le cycle $c = (q_{\times_1}, Y_1, q_{\times_1})$ est lié à p et qu'il ne contient pas de noeud avec un futur vide.

Chaque bMSC de $H3$ est déjà un atome. Donc, nous pouvons associer à p l'expression $E_p = X_1.(Y_1)^*.X_2$.

Nous définissons maintenant deux termes caractérisant un cycle :

Définition 3.7 (Cycle décomposable et null-match)
Soit $E = X_1 \dots X_k.(Y_1 \dots Y_p)^*.W_1 \dots W_q$ *une expression régulière sur un ensemble de bMSCs* $\{X_i\}_{i\in 1..k} \cup \{Y_i\}_{i\in 1..p} \cup \{W_i\}_{i\in 1..q}$. *Nous dirons que* $Y_1 \dots Y_p$ *est* décomposable *dans* E *si* $\exists j$, *tel que* $Futur(X_1 \dots X_k.Y_1 \dots Y_j) = M_\epsilon$. *Nous dirons que* $Y_1 \dots Y_p$ *est* null-match *si* $Futur(X_1 \dots X_k) = M_\epsilon$ *et* $Futur(X_1 \dots X_k.Y_1 \dots Y_p) = M_\epsilon$.

Autrement dit, un cycle est décomposable s'il contient un noeud avec un futur vide, et un cycle est null-match s'il "part" d'un noeud avec un futur vide.

L'idée directrice de l'algorithme qui suit est de réécrire ces expressions à des permutations d'atomes indépendants près pour que les cycles deviennent *décomposable* ou *null match*. Ces nouvelles expressions peuvent alors être utilisées pour calculer un nouveau HMSC équivalent sans chemins problématiques. Considérons à nouveau le HMSC $H3$ et l'expression de coupe de la figure 3.14. L'expression $E_p = X_1.(Y_1)^*.X_2$ est équivalente à l'expression $E_p = (Y_1)^*.X_1.X_2$ pour la quel le chemin $p = (q_{\times_0}, X_1, q_{\times_1})(q_{\times_1}, X_2, q_{\times_2})$ n'est plus problématique.

Nous utiliserons plusieurs types de permutations. Soit $E = X_1.X_2 \dots X_k$ une expression régulière sur un ensemble de bMSCs. Une permutation simple consiste à intervertir, quand c'est possible, deux bMSCs, c'est-à-dire que E peut être réécrit en une expression régulière équivalente $E' = X_1.X_2 \dots X_{i+1}.X_i \dots X_k$ à chaque que fois que X_{i+1} et X_i sont indépendants. En effet, quand deux bMSCs M et N sont indépendants, alors $M \bullet N = N \bullet M$. Cette propriété s'étend à toute séquence de bMSCs. De la même manière, si E contient des cycles, un cycle c peut être déplacé vers l'avant (ou l'arrière) dans une chaîne si tous les bMSCs que c contient sont indépendants de tous les bMSCs que c dépasse. Enfin, deux cycles peuvent être permutés s'ils sont indépendants.

Quand deux expressions régulières E et E' sont équivalentes, nous noterons $E \simeq E'$. Pour une expression E donnée sur un ensemble d'atomes Σ_{At}, nous noterons par $[E]$ sa classe d'équivalence, c'est-à-dire le plus petit ensemble d'expressions sur Σ_{at} contenant E tel que $\forall e \in [E], \forall e' \simeq e, e' \in [E]$. En plus des permutations, nous utilisons des propriétés classiques des expressions régulières telles que $a(ba)^*bc = (ab)^*abc = ab(ab)^*c$ pour trouver la classe d'équivalence d'une expression E. Finalement, quand deux expressions sont équivalentes, alors leurs itérations sont également équivalentes. Notons que pour une expression E, puisque la définition d'équivalence ne déplie jamais une expression étoilée, $[E]$ est fini.

Définition 3.8 (Quotient et Complément) *Soient* E_1, E_2 *deux expressions régulières sur un alphabet* Σ *de bMSCs, générant des mots dans les deux langages* $L_1 \subseteq \Sigma^*$ *et* $L_2 \subseteq \Sigma^*$. *Le* quotient gauche *de* E_1 *par* E_2 *est l'expression régulière* E_1/E_2 *qui génère les mots de* $L_1/L_2 = \{w \in \Sigma^* | \exists v \in L_2 \wedge w.v \in L_1\}$.

Le quotient droit *de* E_1 *par* E_2 *est l'expression régulière* $E_1 \backslash E_2$ *qui génère les mots de* $L_1 \backslash L_2 = \{w \in \Sigma^* | \exists v \in L_2 \wedge v.w \in L_1\}$.

Le complément *d'une expression E est une expression $\overline{E}$ qui génère les mots de* $\Sigma^* - L_E$.

Ces définitions de quotient et de compléments sont utiles pour la définition de remplacement d'une sous-expression par une autre, qui nous est utile pour le remplacement de chemins problématiques.

Définition 3.9 *Soient $H = (\mathcal{N}, T, q_0, \mathcal{Q}_{end}, \mathcal{M})$ un HMSC, E une expression régulière sur $\mathcal{M}^*$, et E' une expression régulière sur $\Sigma = \mathcal{M} \cup At(\mathcal{M})$. Le HMSC obtenu en remplaçant E par E' dans H sera noté $H_{E|E'}$, et il est le HMSC H_{Enew} associé à l'expression régulière*

$$Enew = \big(E_H/(E.\mathcal{M}^*)\big).E'.\big(E_H\backslash(\mathcal{M}^*.E)\big) \cup \big(E_H \cap \overline{\mathcal{M}^*.E.\mathcal{M}^*}\big)$$

Cette définition de remplacement peut être utilisée pour remplacer une expression E non décomposable par une expression décomposable E' à chaque fois que E apparaît dans H. Nous pouvons aussi utiliser cette définition pour remplacer uniquement une occurrence spécifique d'une expression E_p correspondant à un générateur problématique p. Pour faire cela, il est suffisant d'étiqueter de manière différente le bMSC associé à chaque transition (c'est-à-dire s'assurer que $\forall t, t' \in T, \lambda(t) \neq \lambda(t')$ même si les bMSCs sont isomorphes). Avec cette convention, le remplacement d'une expression E_p réécrit un unique chemin de $H_\times$ et ses cycles problématiques liés.

Notons que pour tout générateur problématique p d'un HMSC $H_\times$, puisque toutes les expressions e dans $[E_p]$ génèrent les mêmes scénarios, tous les HMSCs obtenus en remplaçant une expression $e \in [E_p]$ par une autre expression $e' \in [E_p]$ sont équivalents.

Pour un aspect comportemental $A = (P, Ad)$, et un HMSC $H_\times$ avec contextes et futurs obtenus à partir un HMSC H et l'expression de coupe P, nous calculons l'ensemble $\mathcal{P}_u(H_\times)$ de générateurs problématiques de $H_\times$, et l'ensemble P_{exp} des expressions associées à ces générateurs. Alors, si pour tout $E_p \in P_{exp}$, il y a une expression $e \in [E_p]$ dans laquelle toutes les sous-expressions étoilées de e sont soit décomposable soit null match, alors il existe un HMSC H' équivalent à $H_\times$ qui ne contient pas de chemin infini passant indéfiniment souvent par des futurs non vides. Si ce n'est pas le cas, le processus de détection ne peut pas être accompli. Finalement, quand H' peut être calculé, chaque expression E_p relative à un générateur problématique p est remplacée par une des expressions décomposables ou null match trouvée.

Le remplacement des générateurs problématiques est implémenté par l'algorithme 6.

Algorithme 6 Permutations($H_\times, \mathcal{UC}_{H_\times}$)

entrées : un HMSC $H_\times$ avec contextes et futurs, et $\mathcal{UC}_{H_\times}$, l'ensemble des cycles problématiques

sortie : un HMSC H' sans générateur problématique

1: Calcul de $\mathcal{P}_u(H_\times)$, l'ensemble des générateurs problématiques
2: $P_{exp} = \{E_p | p \in \mathcal{P}_u(H_\times)\}$
3: Calcul de E_H, expression régulière associée à H
4: **pour tout** $E_p \in P_{exp}$ **faire**
5: Calcul de $[E_p]$.
6: **si** $\nexists e \in [E_p]$ tel que $\forall c$ lié à e, c est décomposable ou null match **alors**
7: Détection ne peut pas être accomplie ; STOP, retour *nul*
8: **sinon**
9: choisir e des cycles décomposables ou null match
10: $Rep = Rep \cup \{(E_p, e)\}$
11: **fin si**
12: **fin pour**
13: **pour tout** $(E_p, e) \in Rep$ **faire**
14: `/* remplace` E_p `par` e `dans` E_H `*/`
15: $E_H = \begin{array}{l}\big(E_H/(E_p.\mathcal{M}^*)\big).e.\big(E_H \backslash (\mathcal{M}^*.E_p)\big) \\ +\big(E_H \cap \overline{\mathcal{M}^*.E_p.\mathcal{M}^*}\big)\end{array}$
16: **fin pour**
17: $H' = H_{E_H}$ `/* calcul le HMSC équivalent à l'expression` E_H`*/`
18: retour H'

Notons qu'il existe des générateurs problématiques pour lesquels il n'existe pas d'-expression équivalente contenant des cycles décomposables ou null match. Considérons par exemple le HMSC H_4 et l'expression de coupe P_4 de la figure 3.15. Le chemin $p = (q_{\times_0}, X_1, q_{\times_1})(q_{\times_1}, Y, q_{\times_2})\ (q_{\times_2}, X_2, q_{\times_3})$ est clairement un générateur problématique. Les bMSCs X_1 et X_2 sont des atomes et le bMSC Y peut être découpé en deux atomes Y_a et Y_b où Y_a est un bMSC avec uniquement le message "a" sur l'instance $I1$ et Y_b est un bMSC avec uniquement un message "b" sur l'instance $I3$. Une expression associée au chemin p est $E_p = X_1.Y_a.Y_b.(Y_a.Y_b)^*.X_2$. La classe d'équivalence de E_p contient ni de cycles décomposable ni de cycles null match. La seule manière d'obtenir un futur vide est de considérer l'expression $E'_p = (Y_b)^n.X_1.X_2.(Y_a)^n$. Cependant, $E'_p \notin [E_p]$, et même si elle est équivalente à E_p, E'_p n'est pas une expression régulière. Donc, le HMSC de la figure 3.15 ne peut pas être transformé en un HMSC équivalent pour éviter les générateurs problématiques.

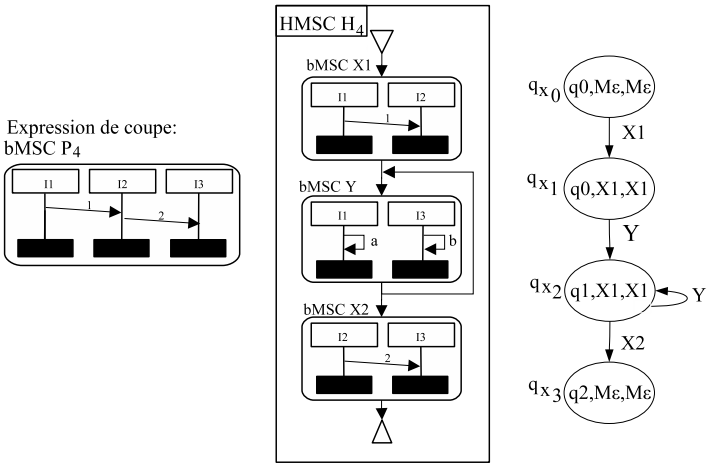


FIG. 3.15 – Détection impossible

3.4.5 HMSC H' et ensemble fini de chemins dans lesquels une expression de coupe est détectée

Pour une expression de coupe P et un HMSC H, si les algorithmes 4, 5 et 6 sont appliqués et s'ils se terminent avec succès, nous obtenons un HMSC H' tel que tous les cycles passent par au moins un noeud avec un futur vide. Le but de l'algorithme 7 est de construire un ensemble fini $\mathcal{P}_m$ de chemins de H' tel que $\forall p \in \mathcal{P}_m$, l'expression de coupe est détectée au moins une fois dans $\lambda(p)$ et tel qu'il n'existe pas de détection partielle de l'expression de coupe dans $\lambda(p)$. $\mathcal{P}_m$ contient alors tous les points de jonction qui peuvent apparaître dans H'.

Plus exactement, $\mathcal{P}_m$ est l'ensemble des chemins d'un HMSC H' tel que pour chaque chemin p de $\mathcal{P}_m$:

i) p part d'un noeud avec un futur vide (le futur est égal à M_ϵ, il n'y donc pas de détection débuter sur un chemin précédent p qui puisse se terminer dans p),

ii) p arrive sur un noeud avec un futur vide (le futur est égal à M_ϵ, il n'y a donc pas de détection qui débute à la fin de p),

iii) tous les autres noeuds de p ont un futur différent du futur vide,

iv) $\exists k \in \mathbb{N}^*$ tel que P est détecté exactement k fois dans le bMSC $\lambda(p)$ associé à p.

Après l'algorithme 6, nous savons que chaque cycle participant à une détection passe par au moins un noeud avec un futur vide. Donc, pour un HMSC H' obtenu en utilisant les algorithmes 4, 5 et 6, chaque chemin de $\mathcal{P}_m$ est acyclique, et $\mathcal{P}_m$ est fini. La construction de $\mathcal{P}_m$ est effectuée par l'algorithme 7.

Algorithme 7 Construction$\mathcal{P}_m$:(P,H')

entrées : expression de coupe P, HMSC $H' = (\mathcal{N}_{tmp}, T_{tmp}, q_{0_{tmp}}, \mathcal{Q}_{end_{tmp}}, \mathcal{M}_{tmp})$.
sortie : $\mathcal{P}_m$ ensemble des chemins dans lesquels P est détecté

1: $\mathcal{P}_m = \{(q_\times, M, q'_\times) \in T_{tmp} \mid F(q_\times) = F(q'_\times) = M_\epsilon \wedge \exists k \in \mathbb{N}, P \triangleright^k M\}$
2: $Chemin = \{(q_\times, M, q'_\times) \in T_{tmp} \mid F(q_\times) = M_\epsilon \wedge F(q'_\times) \neq M_\epsilon\}$
3: **tant que** $Chemin \neq \emptyset$ **faire**
4: $\quad CheminFutur = \emptyset$
5: $\quad$ **pour tout** $p = t_1.t_2...t_k \in Chemin$ **faire**
6: $\quad\quad$ **pour tout** $(q_\times, M, q'_\times) \in T_{tmp}$ tel que $q_\times = \lambda_+(t_k)$ **faire**
7: $\quad\quad\quad$ **si** $F(q'_\times) = M_\epsilon$ **alors**
8: $\quad\quad\quad\quad \mathcal{P}_m = \mathcal{P}_m \cup (p.t)$
9: $\quad\quad\quad$ **sinon**
10: $\quad\quad\quad\quad CheminFutur = CheminFutur \cup (p.t)$
11: $\quad\quad\quad$ **fin si**
12: $\quad\quad$ **fin pour**
13: $\quad$ **fin pour**
14: $\quad Chemin = CheminFutur$
15: **fin tant que**
16: retour $\mathcal{P}_m$

3.5 Limites

Les algorithmes décrits en section 3.4 ont certaines limites. La détection de points de jonction dans un HMSC n'a pas toujours une solution (finie) : pour des HMSCs et des expressions de coupe donnés, il est parfois impossible de construire un HMSC dans lequel le processus de détection est possible. Cette section identifie des conditions suffisantes sur les expressions de coupe et les HMSCs pour que les algorithmes présentés dans ce chapitre terminent.

La première limite de notre approche est liée à la terminaison de l'algorithme 4. Pour un HMSC H, l'algorithme 4 construit un HMSC H_+ où les noeuds de H sont

transformés en des noeuds contenant des contextes (une détection potentielle). Puisque le nombre de noeuds d'un HMSC est fini, l'algorithme termine si et seulement si le nombre de contextes est fini.

Lorsque l'on applique l'algorithme 4 au HMSC H de la figure 3.8, on obtient un nouveau HMSC H_+ qui ne contient que deux contextes différents : M_ϵ et PM. Considérons maintenant le HMSC H_5 et l'expression de coupe P_5 de la figure 3.16. Pour ce cas, la taille des contextes augmente indéfiniment, et il est impossible de construire un HMSC fini avec un ensemble fini de contextes. A partir du noeud (q_0, M_ϵ), la détection potentielle obtenue dans $M_\epsilon \bullet G$ contient un unique message "a" (bMSC $A1$). A partir du noeud $(q_0, A1)$, la détection potentielle obtenue dans $A1 \bullet G$ contient deux messages "a" (bMSC $A2$), et à partir du noeud (q_0, An), (An contient n messages "a"), la détection potentielle obtenue dans $An \bullet G$ contient $n+1$ messages "a" (bMSC $An+1$). A partir d'un noeud et d'un contexte, il est toujours possible de trouver une extension dans laquelle un contexte va être utilisé pour former un point de jonction, mais cette extension crée un nouveau et plus grand contexte que précédemment.

Notons que ce problème n'est pas intrinsèque à l'algorithme, il est réellement insoluble : pour le HMSC H_5 et l'aspect $A = (P_5, Ad_5)$, le tissage attendu est un ensemble de comportements contenant des scénarios de la forme $Ad_5^n.A1^n, n \in \mathbb{N}$. Or cette expression n'est pas une expression régulière sur un ensemble de bMSCs, et elle ne peut pas être représentée par un HMSC.

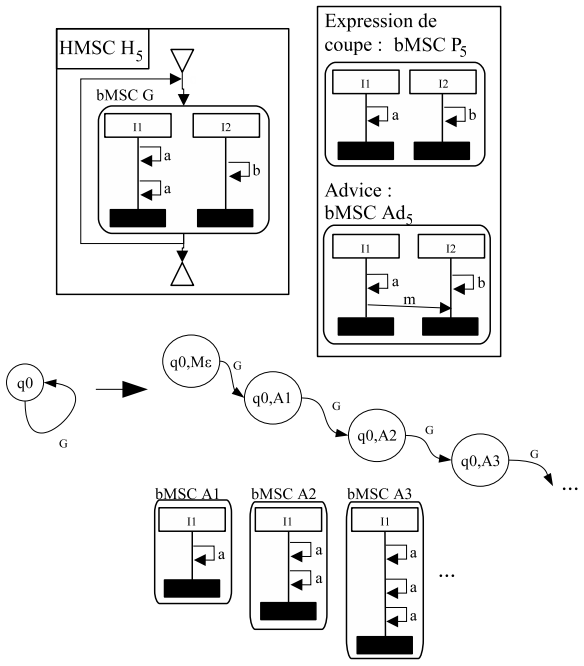


FIG. 3.16 – Un exemple où l'algorithme 4 ne termine pas.

Pour éviter le problème du nombre infini de contextes, nous utiliserons uniquement des bMSCs connexes en tant qu'expression de coupe. Dans ce cas, le nombre de détec-

tions potentielles est fini. Un bMSC M est *connexe* si et seulement si il n'existe pas deux bMSCs $M1$ et $M2$ (différent de M_ϵ) tel que $M = M1 \bullet M2$ et $M1||M2$. Par exemple, dans la figure 3.16, le bMSC A_5 est connexe, tandis que le bMSC P_5 n'est pas connexe ($P_5 = Pa \bullet Pb$ avec Pa un bMSC avec uniquement un message "a" et Pb un bMSC avec uniquement un message "b", et $Pa||Pb$).

Une propriété intéressante est que pour un HMSC H et une expression de coupe P, si P est connexe, alors le nombre de détections potentielles relatives à H et P est borné.

Théorème 3.1 *Soient un HMSC H et P une expression de coupe connexe. Alors le nombre de détections potentielles relatives à H et P est bornée.*

Pour démontrer ce théorème, nous commençons par donner une preuve informelle à travers un exemple. Ensuite, nous reprenons ces explications, mais dans un cadre plus formel et plus général.

Considérons la figure 3.17 où une expression de coupe P est décrite. Il est clair que pour cette expression de coupe et n'importe quel HMSC H, il n'existe que trois détections potentielles possible notées PM_1, PM_2 et PM_3. Nous rappelons que dans la définition 3.3, nous définissons une *partie potentielle* comme un début de détection possible. Par exemple, la détection potentielle PM_1 contient deux parties potentielles : une, avec les deux premiers messages 1 et 2, notée PP_1 sur la figure 3.17 ; l'autre, avec le dernier message 1, noté PP_2. Pour prouver que le nombre de détections potentielles relatives à un HMSC H et à une expression de coupe connexe P est borné, l'idée principale est de montrer que la taille d'une partie potentielle PP_{i+1} est toujours strictement plus petite qu'une partie potentielle PP_i. Dans notre exemple, la partie potentielle PP_2 contient seulement un message (deux événements) tandis que PP_1 contient deux messages (quatre événements). Puisque la taille des parties potentielles dans une détection potentielle est strictement décroissante, quand une expression de coupe est connexe, le nombre de détection potentielle est borné.

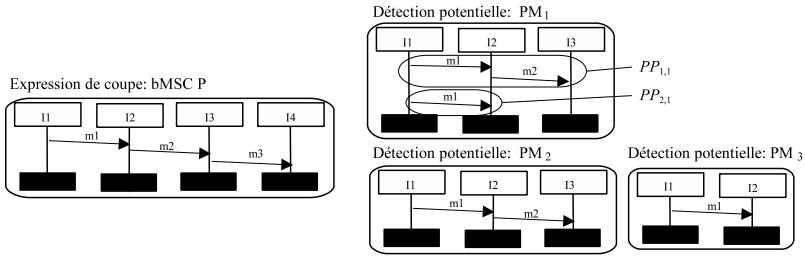


FIG. 3.17 – Un nombre borné de détections potentielles.

Preuve :
De la définition 3.3, nous savons qu'une détection potentielle $P_m = P_{1,1} \bullet P_{2,1} \bullet ... \bullet P_{q,1}$ relative à une expression de coupe P et un bMSC M est composée de q parties potentielles $P_{i,1}, i \in \{1...q\}$. Chaque partie potentielle est nécessairement un préfixe de P.

Premièrement, montrons que $\forall i \in \{1...q-1\}, |P_{i,1}| \geq |P_{i+1,1}|$. Soit e' un événement de $P_{i+1,1}$, tel que $e' \notin P_{i,1}$. Si $\forall e \in P_{i,1}, e||e'$, alors e' devrait appartenir à $P_{i,1}$, car $P_{i,1}$ est

par définition aussi grand que possible : contradiction. Si $\exists e \in P_{i,1} | e \leq e'$ (le cas $e' \leq e$ est impossible, car par définition $P_{i+1,1}$ est localisé après $P_{i,1}$), alors $P_{i,1}$ n'est pas une partie potentielle , car il est impossible de trouver un événement qui joue le même rôle que e' pour $P_{i,1}$: contradiction.

Ensuite, montrons que $\forall i \in \{1...q-1\}, P_{i,1} \neq P_{i+1,1}$. Si $|P| = 1$, la propriété est triviale. Si $|P| > 1$, considérons deux parties potentielles $P_{i,1}$ et $P_{i+1,1}$ tel que $P_{i,1} = P_{i+1,1}$. Puisque P est connexe, $P_{i,1}$ contient au moins un événement e qui a un successeur e' dans P mais tel que e' n'est pas présent dans $P_{i,1}$. Si e appartient aussi à $P_{i+1,1}$, $P_{i,1}$ n'est pas une partie potentielle, car $P_{i+1,1}$ empêche la partie $P_{i,1}$ de participer à un détection éventuelle.

Puisque la taille des parties potentielles dans une détection potentielle est strictement décroissante quand une expression de coupe est connexe, le nombre de détection potentielle est bornée. □

Une autre limite est que le traitement des générateurs problématiques est restreint aux situations où les cycles problématiques sont disjoints. Dans ces situations, les cycles ont les mêmes propriétés que les atomes dans une expression, et les permutations sont plus ou moins des permutations sur des mots finis.

Pour conclure cette section, notons que nous n'avons proposé pour le moment que des conditions suffisantes sur les HMSCs et les expressions de coupe pour que les algorithmes 4 à 7 terminent. La définition de conditions nécessaires permettrait d'identifier des classes du problème de détection pour lesquelles les algorithmes terminent toujours.

3.6 Conclusion et Perspectives

Dans ce chapitre, nous avons présenté un mécanisme de détection de points de jonction (définis en tant que sous-bMSCs) correspondants à une expression de coupe P donnée, dans des scénarios infinis, c'est-à-dire dans des HMSCs de base contenant au moins un cycle. L'intérêt majeur de cette détection est de permettre une détection (et par la suite, un tissage) statique d'aspects dynamiques (comportementaux). L'idée directrice de ce processus réside en la transformation d'un HMSC H de base vers un HMSC H' équivalent, tel qu'il est possible d'identifier un nombre fini de chemins de H' contenant tous les points de jonction relatifs à une expression de coupe. Cette transformation souffre des quelques limitations, mais nous avons donné des conditions suffisantes permettant d'assurer les algorithmes terminent.

Comme travaux futurs, plusieurs améliorations de notre approche peuvent être étudiées. Parmi celles-ci, l'identification d'une classe décidable pour le problème de détection (si elle existe) est une priorité. Une autre amélioration possible est l'utilisation d'autres sémantiques de points de jonction, similaire à celles présentées dans le chapitre précédent (motif, motif sûr, motif clos).

Chapitre 4

Processus de Composition

4.1 Introduction

Dans les deux chapitres précédents, nous avons détaillé un mécanisme de détection de points de jonction. Afin d'obtenir un processus de tissage complet, il nous reste à détailler un mécanisme de composition de la partie d'un aspect que l'on appelle "advice" avec le scénario de base au niveau des points de jonction préalablement détectés.

La composition peut se révéler simple. En effet, lorsqu'un point de jonction est défini à travers la notion de sous-bMSC (comme dans tout le chapitre 3), la composition se résume à un simple remplacement de la partie détectée par l'advice. Nous formalisons ce remplacement dans la section 4.2.

La composition peut également se révéler plus ardue. En effet, lorsque l'on utilise des points de jonction définis à travers les notions de motifs (incluant les motifs clos et sûrs), la composition de l'advice avec les parties détectées n'est plus qu'un simple remplacement. Nous devons considérer les événements d'un point de jonction et les événements qui peuvent être intercalés entre les événements d'un point de jonction, pour fusionner le tout avec le comportement spécifié dans l'advice. Pour cela, nous proposons dans la section 4.3 une définition formelle d'un nouvel opérateur de composition de bMSCs, appelé *somme amalgamée*.

Finalement, nous proposons dans la section 4.4 une composition à un niveau plus abstrait, c'est-à-dire au niveau des HMSCs. Cette composition, appelée *produit fibré*, n'a pas de rapport direct avec les mécanismes de détection proposés dans les chapitres précédents, mais elle pourrait se révéler très utile pour tisser des aspects à un plus haut niveau d'abstraction, comme le montrent les deux applications proposées.

4.2 La Composition vue comme un Remplacement

Pour un bMSC M et un aspect comportemental $A = (P, Ad)$, lorsque les points de jonction sont définis à travers la notion de sous-bMSC, nous rappelons (d'après les définitions 3.1 et 3.2 du chapitre précédent) que si l'expression de coupe P permet la détection de k points de jonction dans M, alors il existe une écriture de M sous la forme

$M = X_1 \bullet P \bullet X_2 \bullet \cdots \bullet P \bullet X_{k+1}$. Pour composer (ou tisser) l'advice Ad avec M au niveau des points de jonction, il suffit de remplacer P par Ad. Nous obtenons alors le tissage de A dans M, noté $M \otimes A$: $M \otimes A = X_1 \bullet Ad \bullet X_2 \bullet \cdots \bullet Ad \bullet X_{k+1}$.

La figure 4.1 illustre un processus de tissage complet : après la détection des points de jonction, le bMSC M peut s'écrire sous la forme $X_1 \bullet P \bullet X_2 \bullet P \bullet X_3 = P \bullet P \bullet X_3$ (car $X_1 = X_2 = M_\epsilon$). Lors de la composition, nous remplaçons P par Ad, et nous obtenons : $M \otimes A = Ad \bullet Ad \bullet X_3$.

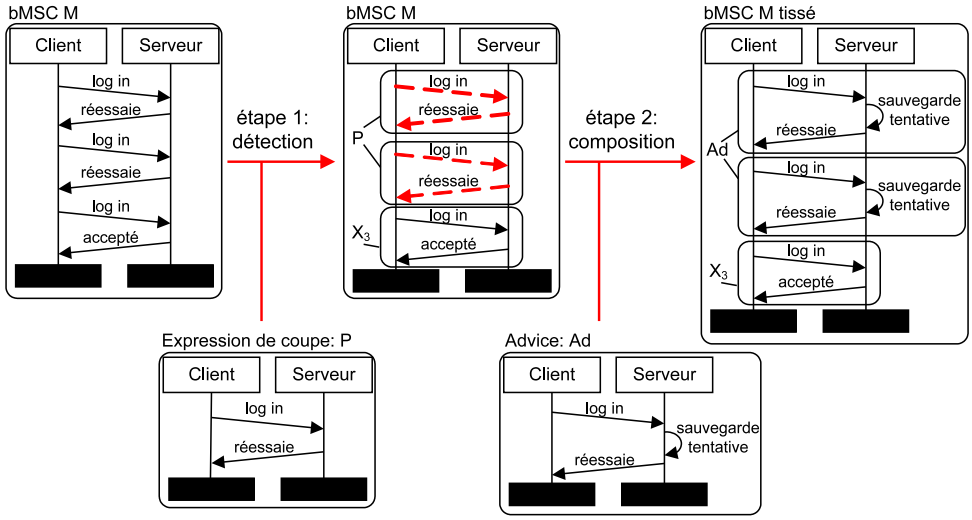


FIG. 4.1 – Illustration du remplacement des points de jonction par l'advice

La détection de points de jonction définis comme des sous-bMSCs dans des comportements infinis a été présentée dans le chapitre précédent. Nous avons détaillé la transformation d'un HMSC de base H en un HMSC équivalent H' et identifié tous les points de jonction dans un ensemble fini $\mathcal{P}_m$ de chemins de H'. Les seules choses qui restent à faire, pour terminer le processus de tissage, sont de remplacer, pour chaque chemin p de $\mathcal{P}_m$, tous les points de jonction présents dans p par l'advice pour obtenir un nouveau chemin p', et de remplacer p par p'. Comme pour les permutations détaillées dans la section 3.4, cela peut se faire à travers la réécriture d'une expression régulière (définition 3.9). L'algorithme 8 décrit ce processus de réécriture qui finalise le tissage d'un aspect comportemental dans des comportements infinis.

4.3 La Composition vue comme une Somme Amalgamée

Nous rappelons que lorsque les points de jonction sont définis comme des sous-bMSCs, si nous notons J un point de jonction, et B un bMSC de base, par définition, nous pouvons écrire : $B = B_1 \bullet J \bullet B_2$. Si Ad est l'advice représentant le comportement attendu, la composition réside uniquement à remplacer J par Ad. Nous obtenons ainsi : $B = B_1 \bullet Ad \bullet B_2$. Cependant, lorsque l'on utilise les notions de motifs, motifs sûrs, ou motifs clos pour définir les points de jonction, la composition de l'advice n'est pas aussi

Algorithme 8 Tissage(H,A)

entrées : un HMSC $H_{=}(\mathcal{N}, T, q_0, \mathcal{Q}_{end}, \mathcal{M})$,
un aspect comportemental $A = (P, Ad)$ et un ensemble de chemins $\mathcal{P}_m$
sortie : un HMSC tissé H'.

1: $E'_H = E_H$
2: **pour tout** $p \in \mathcal{P}_m$ **faire**
3: calcul E_p `/* l'expression régulière associée à` p `*/`
4: `Trouve la factorisation minimale` $x_1 \bullet P \bullet x_2 \cdots \bullet P \bullet x_k$ `de` E_p
5: $E'_p := x_1 \bullet Ad \bullet x_2 \cdots \bullet Ad \bullet x_k$
6: `/* remplace` E_p `par` E'_p `dans` E'_H `*/`
7: $E'_H = \begin{array}{l} \left(E'_H/(E_p.\mathcal{M}^*)\right).E'_p.\left(E'_H \backslash (\mathcal{M}^*.E_p)\right) \\ +\left(E'_H \cap \overline{\mathcal{M}^*.E_p.\mathcal{M}^*}\right) \end{array}$
8: **fin pour**
9: retour H'

simple. En effet, avec ces types de points de jonction, des messages peuvent croiser les points de jonction ou être intercalés entre les messages constituant les points de jonction. Par exemple, dans le bMSC M de la figure 4.2, le message "sauvegarde tentative" est intercalé entre les messages "log in" et "réessaie" qui forment un point de jonction associé à l'expression de coupe P. Dans ce genre de situation, il n'est pas possible de simplement remplacer le point de jonction par l'advice, car le résultat escompté ne peut pas toujours être exprimé avec des opérateurs de composition standards tels que la composition séquentielle. Pour cette raison, nous devons définir de nouveaux opérateurs de composition qui prennent en compte les parties communes entre le point de jonction et l'advice (les messages "log in" et "réessaie" et les instances "client" et "serveur" dans notre exemple de la figure 4.2), ainsi que les éléments non communs (comme le message "mise à jour" et l'instance "écran") pour produire un nouveau bMSC qui ne contient pas de copie d'éléments similaires aux deux opérandes.

Dans cette section, nous présentons deux opérateurs de compositions de bMSCs appelés *somme amalgamée* et *somme amalgamée gauche*. Nous commençons par présenter la somme amalgamée qui est un opérateur permettant de composer deux vues d'un même système tout en tenant compte des éléments communs à ces deux vues. Cet opérateur n'est pas directement adapté à l'utilisation des aspects. Pour cette raison, nous présentons également la somme amalgamée gauche, qui est une variante de la somme amalgamée, mais adaptée aux aspects.

4.3.1 Somme Amalgamée

Les compositions usuelles liées aux bMSCs sont limitées aux compositions séquentielles et parallèles, à l'itération et à l'alternative. D'autres opérations sur les bMSCs ont été proposées, telles que le raffinement des instances [MR96], le raffinement des messages

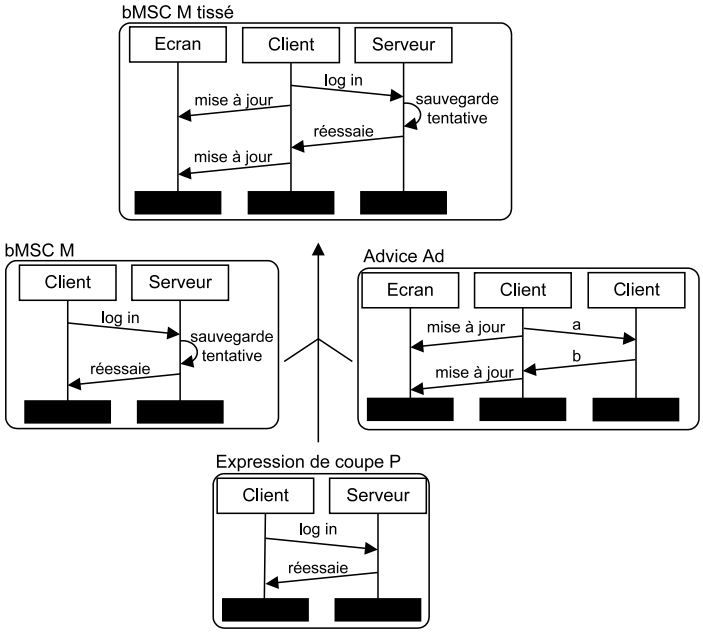


FIG. 4.2 – Un exemple de Composition

[Eng98], la virtualité [RGG95], ou plus récemment la projection [GHM03][1]. Cependant, quand deux bMSCs décrivent différents points de vue d'un même comportement, on ressent la nécessitée d'un opérateur de fusion qui composerait les deux scénarios pour produire un résultat qui contient les deux opérandes sans créer de copie d'éléments similaires. Cet opérateur ne peut pas être exprimé à l'aide des opérateurs classiques. Nous proposons un opérateur de fusion de bMSCs appelé *somme amalgamée*. Le terme somme amalgamée provient des concepts de la théorie des catégories. Cependant, dans ce chapitre, nous n'y ferons pas explicitement référence, préfèrent certains concepts de la théorie des ensembles. Plus de détails sur la théorie des catégories pourront être trouvés dans [Lan98].

Avant de définir une somme amalgamée, nous rappelons la notion de morphismes de bMSCs, qui est essentielle pour définir les parties communes de deux scénarios.

Définition 4.1 (Morphisme de bMSCs) *Soit deux bMSCs* $M = (I, E, \leq, A, \alpha, \phi, \prec)$ *et* $M' = (I', E', \leq', A', \alpha', \phi', \prec')$. *Un morphisme de bMSCs, de* M *vers* M' *est un triplet* $\mu =< \mu_0, \mu_1, \mu_2 >$ *de morphismes, où* $\mu_0 : I \to I'$, $\mu_1 : E \to E'$ *est injective,* $\mu_2 : A \to A'$ *est une fonction de renomage et :*

$$(i)\ \ \forall (e,f) \in E^2, e \leq f \Rightarrow \mu_1(e) \leq' \mu_1(f) \qquad (iii)\ \ \mu_0 \circ \phi = \phi' \circ \mu_1$$
$$(ii)\ \ \forall (e,f) \in E^2, e \prec f \Rightarrow \mu_1(e) \prec' \mu_1(f) \qquad (iv)\ \ \mu_2 \circ \alpha = \alpha' \circ \mu_1$$

[1] Il existe également, par exmple dans UML2.0, d'autres opérateurs plus "exotiques" mais qui n'ont pas de réelles sémantiques.

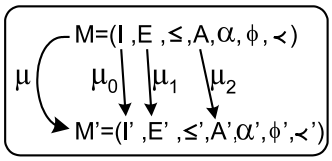


FIG. 4.3 – Illustration des ensembles de départ et d'arrivée d'un morphisme de bMSCs

La figure 4.3 illustre les ensembles de départ et d'arrivée des morphismes constituant un morphisme de bMSCs.

Notons que les propriétés (i) et (ii) garantissent que par un morphisme de bMSCs l'ordre des événements est préservé ainsi que la nature des événements (par exemple, un envoi de message de M sera toujours associé à un envoi de message de M'). La propriété (iii) signifie que tous les événements localisés sur un objet de M sont envoyés sur un seul objet de M'.

Définition 4.2 (Somme Amalgamée de Deux Ensembles) *Soient trois ensembles finis I, J et K. Soient deux fonctions injectives $f : I \to J$ et $g : I \to K$. La somme amalgamée $J\,{}_f+_g\,K$ est définie comme $J\,{}_f+_g\,K = \big(J \backslash f(I)\big) \uplus \big(K \backslash g(I)\big) \uplus I$. La somme amalgamée produit deux injections $\tilde{f} : J \to J\,{}_f+_g\,K$ et $\tilde{g} : K \to J\,{}_f+_g\,K$ définis comme suit :*

$$\begin{cases} \forall i \in f(I), & \tilde{f}(i) = f^{-1}(i) \\ \forall i \in J \setminus f(I), & \tilde{f}(i) = i \end{cases} \qquad \begin{cases} \forall i \in g(I), & \tilde{g}(i) = g^{-1}(i) \\ \forall i \in K \setminus g(I), & \tilde{g}(i) = i \end{cases}$$

Notons que comme nous utilisons $\uplus$ (union disjointe) dans notre définition, le résultat d'une somme amalgamée peut contenir plusieurs copies d'éléments similaires. Les éléments qui ne sont pas dupliqués sont images d'un même élément dans I par f et g.

La figure 4.4 illustre la somme amalgamée des ensembles J et K. Premièrement, conformément à la définition, $J\,{}_f+_g\,K$ contient tous les éléments "communs" à J et K, c'est-à-dire les éléments présents dans I. Par exemple, grâce aux morphismes f et g, comme l'élément a de J et l'élément m de K ont le même antécédent x, on peut remplacer a et m par x dans $J\,{}_f+_g\,K$ (on peut dire que a et m jouent le même rôle). Ensuite, il reste à ajouter dans $J\,{}_f+_g\,K$, les éléments "différents" à J et K, c'est-à-dire tous les éléments de J n'ayant pas d'antécédent par f ainsi que tous les éléments de K n'ayant pas d'antécédent par g (c'est le cas des éléments c, d et r).

La somme amalgamée de deux ensembles sera utilisée pour amalgamer les ensembles d'instances, d'événements et d'actions de deux bMSCs.

Définition 4.3 (Somme Amalgamée de deux bMSCs) *Soient trois bMSCs $M_0 = (I_0, E_0, \leq_0, A_0, \alpha_0, \phi_0, \prec_0)$, $M_1 = (I_1, E_1, \leq_1, A_1, \alpha_1, \phi_1, \prec_1)$, $M_2 = (I_2, E_2, \leq_2, A_2, \alpha_2, \phi_2, \prec_2)$ et deux bMSCs morphismes $f =< f_0, f_1, f_2 >: M_0 \to M_1$, $g =< g_0, g_1, g_2 >: M_0 \to M_2$. La somme amalgamée de M_1 et M_2 sous f et g est le bMSC $M = M_1\,{}_f+_g\,M_2$ où $M = (I, E, \leq, A, \alpha, \phi, \prec)$ est définie par :*

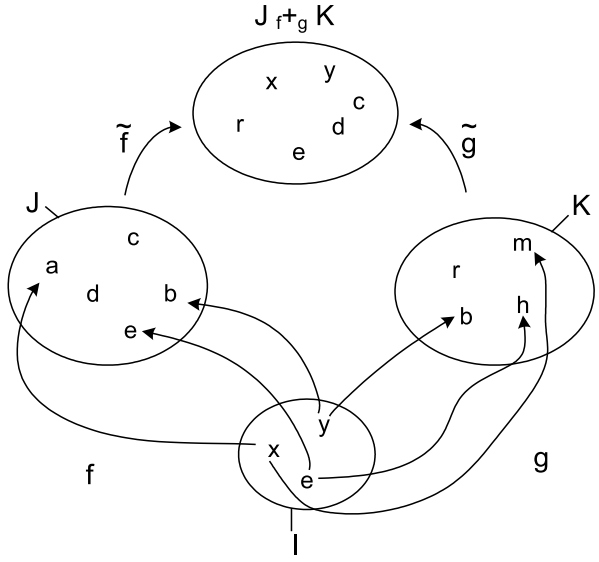


FIG. 4.4 – Une somme amalgamée des deux ensembles J et K

- $I = I_1 \,{}_{f_0}+_{g_0}\, I_2$; $\quad E = E_1 \,{}_{f_1}+_{g_1}\, E_2$; $\quad A = A_1 \,{}_{f_2}+_{g_2}\, A_2$;
- *la relation d'ordre* $\leq$ *est la fermeture transitive de* $\tilde{f}_1(\leq_1) \cup \tilde{g}_1(\leq_2)$;
- $\forall e \in E, \alpha(e) = \begin{cases} \alpha_1(e) \ si\ e \in E_1 \backslash f_1(E_0) \\ \alpha_2(e) \ si\ e \in E_2 \backslash f_2(E_0) \\ \alpha_0(e) \ sinon \end{cases}$, $\quad \phi(e) = \begin{cases} \phi_1(e) \ si\ e \in E_1 \backslash f_1(E_0) \\ \phi_2(e) \ si\ e \in E_2 \backslash f_2(E_0) \\ \phi_0(e) \ sinon \end{cases}$
- $\prec = \tilde{f}_1(\prec_1) \cup \tilde{g}_1(\prec_2)$.

Le bMSC M_0 *est appelé l'*interface *de la somme amalgamée* $M_1 \,{}_f+_g\, M_2$.

Illustrons l'utilisation de la somme amalgamée grâce à l'exemple de la figure 4.5. Considérons les deux bMSCs $M_1 = (I_1, E_1, \leq_1, A_1, \alpha_1, \phi_1, \prec_1)$ et $M_2 = (I_2, E_2, \leq_2, A_2, \alpha_2, \phi_2, \prec_2)$ comme deux vues partielles d'un même système. Nous voulons obtenir le comportement qui contient M_1 et M_2. Supposons que même si M_1 et M_2 ont des ensembles différents d'instances, l'instance X dans M_2 et l'instance *sender* dans M_1 (ou Y et *medium*) représentent le même objet dans le sytème. De manière intuitive, fusionner M_1 et M_2 consiste à insérer le message interne m_3 entre l'émission *data* et la réception *ack* de M_1, ainsi qu'à renommer les instances. Formellement, la fusion consiste à définir une "interface" qui identifie les éléments communs à M_1 et M_2, et les renommer. Dans notre exemple, cela est fait en utilisant un nouveau bMSC $M_0 = (I_0, E_0, \leq_0, A_0, \alpha_0, \phi_0, \prec_0)$, et deux morphismes de bMSCs $f : M_0 \to M_1$ et $g : M_0 \to M_2$ définis comme suit :

- le morphisme $f : M_0 \to M_1$ est un triplet $f =< f_0, f_1, f_2 >$, où :
  - $f_0 : I_0 \to I_1$ est l'identité,
  - $f_1 : E_0 \to E_1$ envoie respectivement $e_{01}, e_{02}, e_{03}, e_{04}$ vers $e_{11}, e_{12}, e_{13}, e_{15}$,

- $f_2 : A_0 \rightarrow A_1$ est l'identité.

– le morphisme $g : M_0 \rightarrow M_2$ est un triplet $g =< g_0, g_1, g_2 >$, où :
  - $g_0 : I_0 \rightarrow I_2$ envoie *sender* vers X et *medium* vers Y,
  - $g_1 : E_0 \rightarrow E_2$ envoie respectivement $e_{01}, e_{02}, e_{03}, e_{04}$ vers $e_{21}, e_{24}, e_{25}, e_{26}$,
  - $g_2 : A_0 \rightarrow A_2$ envoie respectivement $!data, ?data, !ack, ?ack$ vers $!m1$, $?m1$, $!m2$, $?m2$.

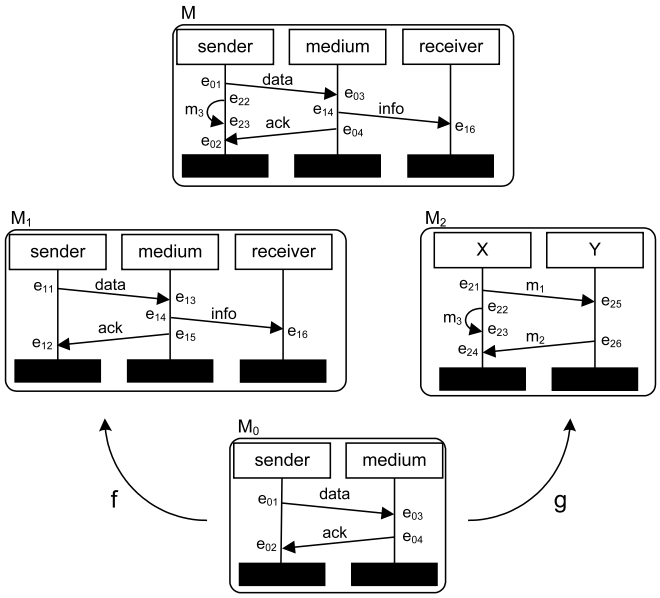


FIG. 4.5 – Un exemple de somme amalgamée de deux bMSCs

Le résultat de la somme amalgamée $M_1 \ {}_f +_g \ M_2$ est le bMSC M.

Notons que les noms des éléments communs résultants, par exemple ceux des instances, sont ceux définis sur l'interface. De cette manière, la somme amalgamée peut également servir à renommer des instances, des messages, etc. Ce mécanisme peut trouver une utilité dans la définition et l'instanciation de comportements génériques.

Notons qu'une somme amalgamée de deux bMSCs peut engendrer des corégions. Comme définie dans le chapitre 1, une corégion est une zone de l'axe d'une instance, représentée par des pointillés, dans laquelle les événements ne sont pas ordonnés. Par exemple, le bMSC M de la figure 4.6 montre une corégion sur l'instance *medium* : les événements e_{14} et e_{23} précédent l'événement e_{04} et succèdent l'événement e_{03}. Par contre, les événements e_{14} et e_{23} ne sont pas ordonnés. Le bMSC M de la figure 4.6 est obtenu en amalgamant le bMSC M_1 et le bMSC M_2 de la même figure (nous ne détaillons pas cette somme amalgamée car elle est très proche de celle que l'on vient de présenter). Dans cet exemple, Nous obtenons une corégion dans le bMSC M, car conformément à la définition de la somme amalgamée, il est impossible d'ordonner l'envoi des messages m_3 et $info$.

Notons également qu'une somme amalgamée de deux bMSCs bien formés M_1 et M_2 n'est pas toujours un bMSC bien formé, car l'ordre contenant $\leq_1$ et $\leq_2$ peut ne pas

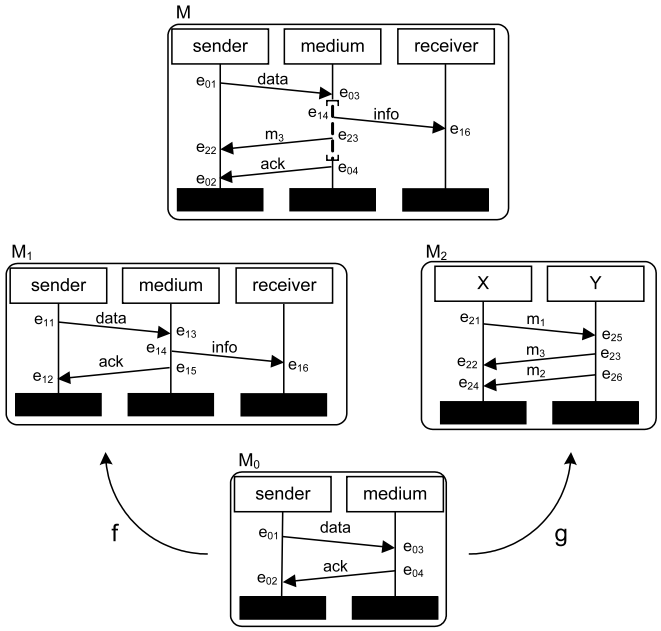


FIG. 4.6 – Un exemple de corégion

être antisymétrique. Considérons l'exemple de la figure 4.7. Le bMSC M_1 impose que $e_{11} \leq_1 e_{15}$, tandis que le bMSC M_2 impose que $e_{24} \leq_2 e_{22}$. De manière évidente, si e_{11} et e_{15} sont respectivement identifié à e_{22} et e_{24} grâce à l'interface, la somme amalgamée de M_1 et M_2 crée un cycle dans les dépendances causales, qui peut facilement être détectée. Dans un tel cas (avec l'identification des éléments communs proposés), les deux scénarios sont incompatibles. Dans [HHC06], Hélouët et al. montrent que la vérification de la compatibilité de deux bMSCs amalgamés, revient à trouver les composantes fortement connexes d'un graphe. Ce qui peut se faire en temps polynomial grâce à l'algorithme de Tarjan [Tar72].

Notons finalement que l'un des inconvénients d'une somme amalgamée de deux bMSCs est que le concepteur doit fournir explicitement la description des parties communes des deux opérandes (il doit spécifier l'interface et deux morphismes de bMSCs). Dans [HHC06], Hélouët et al. proposent une construction automatique de l'interface. Plusieurs interfaces peuvent être considérées comme candidates pour la somme. Pour cette raison, les auteurs proposent également une heuristique qui permet de trouver efficacement les "meilleures " interfaces parmi l'ensemble des interfaces possibles.

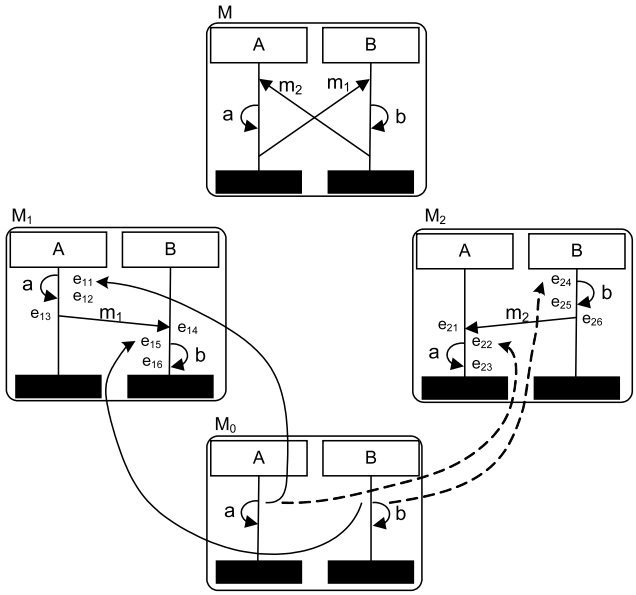


FIG. 4.7 – Une somme amalgamée dans le résultat est mal formé

4.3.2 Somme Amalgamée Gauche

La somme amalgamée définie dans la sous-section précédente présente trois inconvénients pour le tissage d'aspects. Premièrement, ce sont les noms présents dans l'interface d'une somme amalgamée qui sont préservés, alors que lorsque l'on tisse un aspect dans un bMSC de base M ce sont les noms présents dans M que l'on veut préserver. Deuxièmement, la somme amalgamée ne permet pas de supprimer des éléments. En effet, si l'on somme un bMSC M_1 avec un bMSC M_2, tous les éléments de M_1 et de M_2 (au nommage des éléments près) sont préservés. Or, il nous paraît utile de tisser des aspects qui puissent supprimer des comportements d'un bMSC de base. Troisièment, l'ordre sur les événements obtenu après une somme amalgamé ne nous convient pas totalement. Pour ces raisons, nous proposons une variante de la somme amalgamée, appelée *somme amalgamée gauche*.

Dans la suite de cette sous-section, nous commençons par présenter la somme amalgamée gauche de deux ensembles, qui nous servira à la définition de la somme amalgamée gauche de deux bMSCs.

4.3.2.1 Somme Amalgamée Gauche de Deux Ensembles

Voici la définition de la somme amalgamée gauche de deux ensembles :

Définition 4.4 (Somme Amalgamée Gauche de Deux Ensembles) *Soient trois ensembles finis I, J et K. Soient deux fonctions injectives $f : I \to J$ et $g : I' \subseteq I \to K$. La somme amalgamée gauche $J \,{}_f|+_g\, K$ est définie comme :*

$$J \,{}_f|+_g\, K = \big(J \backslash f(I \backslash I')\big) \uplus \big(K \backslash g(I)\big)$$

De façon plus intuitive, lorsque l'on souhaite faire la somme amalgamée gauche de deux ensembles J et K (J étant l'opérande de gauche), on garde tous les éléments de J et l'on rajoute les éléments de K qui ne sont pas en communs avec les éléments de J. Comme dans la somme amalgamée précédente, les éléments en commun sont identifiés grâce à un troisième ensemble et deux morphismes. Pour pouvoir supprimer un élément j de J, nous l'identifions en l'associant avec le morphisme f à un élément i de I tel que i n'a pas d'image par g dans K.

La figure 4.8 montre la somme amalgamée gauche des deux ensembles J et K de la figure 4.4. Nous voyons que dans $J \,|_f + _g K$, nous avons tous les éléments de J plus l'élément r de K qui n'est pas en commun avec J. Les éléments b, h et m de K ont été identifiés comme commun aux éléments respectifs b, e et a de J grâce aux morphismes f et g et à l'ensemble I.

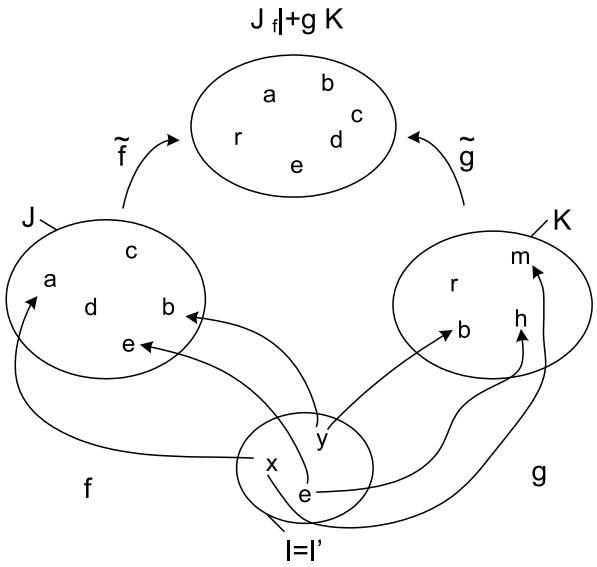


FIG. 4.8 – Un exemple de somme amalgamée gauche

La figure 4.9 montre un autre exemple de somme amalgamée gauche qui ressemble beaucoup à l'exemple précédent à l'exception de l'absence de l'élément m dans K. L'objectif de cet exemple est de montrer une suppression d'éléments de J. Selon la définition de la somme amalgamée gauche, puisque l'élément a de J a un antécédent x dans I qui lui-même n'a pas d'image dans K, a est suprimé dans le résultat. Nous remarquons également que g n'est une injection que de I' vers K.

4.3.2.2 Somme Amalgamée Gauche de Deux bMSCs

La somme amalgamée gauche de deux ensembles sera utilisée pour amalgamer les ensembles d'instances, d'événements et d'actions de deux bMSCs. Le point important qui reste à considérer pour le calcul d'une somme amalgamée gauche de deux bMSCs est le calcul de l'ordre partiel résultant. Mais avant d'expliquer ce point, considérons

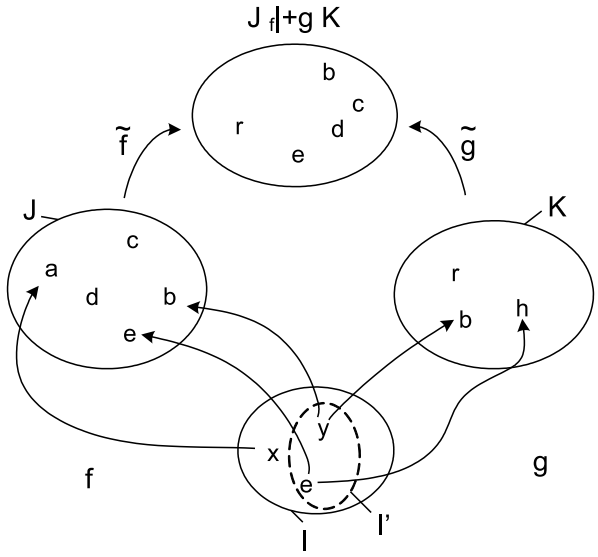
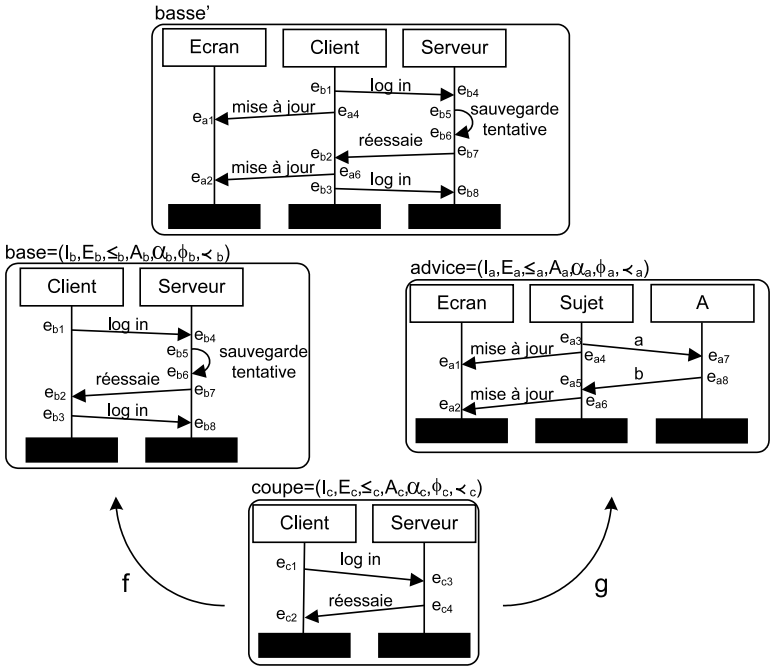


FIG. 4.9 – Un exemple de somme amalgamée gauche avec supression

l'exemple de la figure 4.10. Cette figure montre la somme amalgamée gauche des deux bMSCs $base = (I_b, E_b, \leq_b, A_b, \alpha_b, \phi_b, \prec_b)$ et $advice = (I_a, E_a, \leq_a, A_a, \alpha_a, \phi_a, \prec_a)$. Nous utilisons un troisième bMSC $pointcut = (I_p, E_p, \leq_p, A_p, \alpha_p, \phi_p, \prec_p)$ et deux morphismes de bMSCs $f : pointcut \rightarrow base$ and $g : pointcut \rightarrow advice$ qui permettent la spécification des parties communes aux deux bMSCs *base* and *advice* (et la spécification des éléments à supprimer). Les morphismes f et g sont décrits en détail sur la figure 4.10, et nous pouvons constater que g indique, par exemple, que l'instance *client* joue le rôle de l'instance *sujet* ou que l'instance *serveur* joue le rôle de l'objet A dans le bMSC *advice*.

Dans la figure 4.10, les éléments des bMSCs *base* et *advice* ayant le même antécédent par f et g seront considérés comme identiques dans le bMSC *base'*, mais ils garderont les noms spécifiés dans le bMSC *base*. Par exemple, les instances *sujet* et A dans le bMSC *advice* seront remplacés par les instances *client* et *serveur*. Tous les éléments d'un bMSC *base* ayant un antécédent γ par f tel que γ n'a pas d'image par g dans le bMSC *advice* sont supprimés. Ce cas n'apparaît pas dans l'exemple proposé, mais de cette manière nous pouvons supprimer des éléments du bMSC *base*. Par exemple, dans une somme amalgamée gauche, si l'opérande de droite (le bMSC *advice* dans notre exemple) est un bMSC vide, alors tous les éléments de l'opérande de gauche ayant un antécédent par f dans le bMSC *coupe* sont supprimés. Dans l'exemple de la figure 4.10, si *advice* etait un bMSC vide, le résultat ne comporterait que le message *sauvegarde tentative* et le deuxième message *log in*.

Finalement, tous les éléments des bMSCs *base* et *advice* n'ayant pas d'antécédent par f et g sont gardés dans le bMSC *base'*, mais les événements du bMSC *advice* formeront toujours un "bloc" autour duquel les événements du bMSC *base* seront ajoutés. Dans une somme amalgamée classique (celle de la sous-section précédente), les envoies des messages *mise à jour* et *log in* (correspondants aux événements e_{a6} et e_{b3}) ne seraient



f=(f_0,f_1,f_2): coupe → base, où:
- f_0: I_c → I_b est l'identitée
- f_1: E_c → E_b envoie e_{c1}, e_{c2}, e_{c3} et e_{c4}, respectivement vers e_{b1}, e_{b2}, e_{b4} et e_{b7}
- f_2: A_c → A_b est l'identitée

g=(g_0,g_1,g_2): coupe → advice, où:
- g_0: I_c → I_a envoie Client vers Sujet et Serveur vers A
- g_1: E_c → E_a envoie e_{c1}, e_{c2}, e_{c3} et e_{c4}, respectivement vers e_{a3}, e_{a5}, e_{a7} et e_{a8}
- g_2: A_c → A_a envoie log in vers a et réessai vers b

FIG. 4.10 – Un exemple de somme amalgamée gauche de deux bMSCs

pas ordonnés. Par contre, dans une somme amalgamée gauche, l'envoi du message *mise à jour* du bMSC *advice* s'effectue avant l'envoi du message *log in* du bMSC *base*.

La figure 4.11 donne une vision schématique de l'ordre attendu. Les parties grisées représentent les éléments communs aux deux opérandes. Les parties Pre représentent les événements qui précèdent les éléments communs, tandis que les parties $Post$ représentent les événements qui succèdent aux éléments communs. Les parties In représentent les événements intercalés entre les éléments communs, et les parties *autre* représentent les événements qui ne sont pas ordonnés par rapport aux éléments communs. L'ordre attendu place immédiatement avant et après les éléments communs les parties Pre et $Post$ de l'opérande de droite, et seulement ensuite les parties Pre et $Post$ de l'opérande de gauche. C'est pour cette raison, que nous disons que les événements de l'opérande de droite forment un "bloc" autour duquel les événements de l'opérande de gauche, excepté ceux en communs, sont ajoutés. Parfois, les événements des parties In et *autre* des deux opérandes ne pourront pas être ordonnés. Nous obtiendrons alors des corégions.

Plus formellement, nous pouvons définir la somme amalgamée gauche de deux bMSCs comme suit :

Définition 4.5 (Somme Amalgamée Gauche de deux bMSCs) *Soient trois bMSCs* $M_0 = (I_0, E_0, \leq_0, A_0, \alpha_0, \phi_0, \prec_0)$, $M_1 = (I_1, E_1, \leq_1, A_1, \alpha_1, \phi_1, \prec_1)$, $M_2 = (I_2, E_2, \leq_2, A_2, \alpha_2, \phi_2, \prec_2)$, *et deux morphisms de bMSCs* $f =< f_0, f_1, f_2 >: M_0 \to M_1$ *et* $g =< g_0, g_1, g_2 >: M'_0 \subseteq M_0 \to M_2$. *La somme amalgamée de* M_1 *et* M_2 *est le bMSC* $M = M_1 \,{}_f|\, +_g M_2$ *où* $M = (I, E, \leq, A, \alpha, \phi, \prec)$ *est défini par :*

$I = I_1 \,{}_{f_0}| +_{g_0} I_2$; $\quad E = E_1 \,{}_{f_1}| +_{g_1} E_2$; $\quad A = A_1 \,{}_{f_2}| +_{g_2} A_2$;

Soient $E'_1 = E_1 \cap E$ *l'ensemble des éléments de* E_1 *présents dans* M. *Découpons* E'_1 *en cinq sous-ensembles tel que* $E'_1 = E_{com} \cup Pre \cup Post \cup In \cup Autre$ *où :*

- $E_{com} = f_1(E_0) \cap E$ *est l'ensemble des éléments communs à* E_1 *et* E_2 *(identifiés par les morphismes),*
- $Pre = \{e \in E'_1 \backslash E_{com} | \exists e' \in E_{com}, e \leq_1 e' \wedge \nexists e'' \in E_{com}, e'' \leq_1 e\}$ *est l'ensemble des éléments de* E_1 *qui précédent les éléments de* E_{com},
- $Post = \{e \in E'_1 \backslash E_{com} | \exists e' \in E_{com}, e' \leq_1 e \wedge \nexists e'' \in E_{com}, e \leq_1 e''\}$ *est l'ensemble des éléments de* E_1 *qui succèdent les éléments de* E_{com},
- $In = \{e \in E'_1 \backslash E_{com} | \exists e' \in E_{com}, e' \leq_1 e \wedge \exists e'' \in E_{com}, e \leq_1 e''\}$ *est l'ensemble des éléments de* E_1 *qui sont intercalés entre les éléments de* E_{com}.
- $Autre = \{e \in E'_1 \backslash E_{com} | \nexists e' \in E_{com}, e' \leq_1 e \wedge \nexists e'' \in E_{com}, e \leq_1 e''\}$ *est l'ensemble des éléments de* E_1 *qui sont concurrents aux éléments de* E_{com}.

alors,

$$\leq = \left(\begin{array}{l} \{(e_1, e_2) \in (E'_1)^2 | e_1 \leq_1 e_2\} \cup \\ \{(e_1, e_2) \in (E_2 \cap E)^2 | e_1 \leq_2 e_2\} \cup \\ \{(e_1, e_2), e_1 \in E_{com}, e_2 \in (E_2 \cap E) | \\ \qquad \exists e'_2 \in E_2, e'_2 = g_1 \circ f_1^{-1}(e_1) \wedge e'_2 \leq_2 e_2\} \cup \\ \{(e_1, e_2), e_1 \in (E_2 \cap E), e_2 \in E_{com} | \\ \qquad \exists e'_2 \in E_2, e'_2 = g_1 \circ f_1^{-1}(e_2) \wedge e_1 \leq_2 e'_2\} \cup \\ \{(e_1, e_2), e_1 \in Pre, e_2 \in (E_2 \cap E) | \phi(e_1) = \phi(e_2)\} \cup \\ \{(e_1, e_2), e_1 \in (E_2 \cap E), e_2 \in Post | \phi(e_1) = \phi(e_2)\} \end{array} \right)^*$$

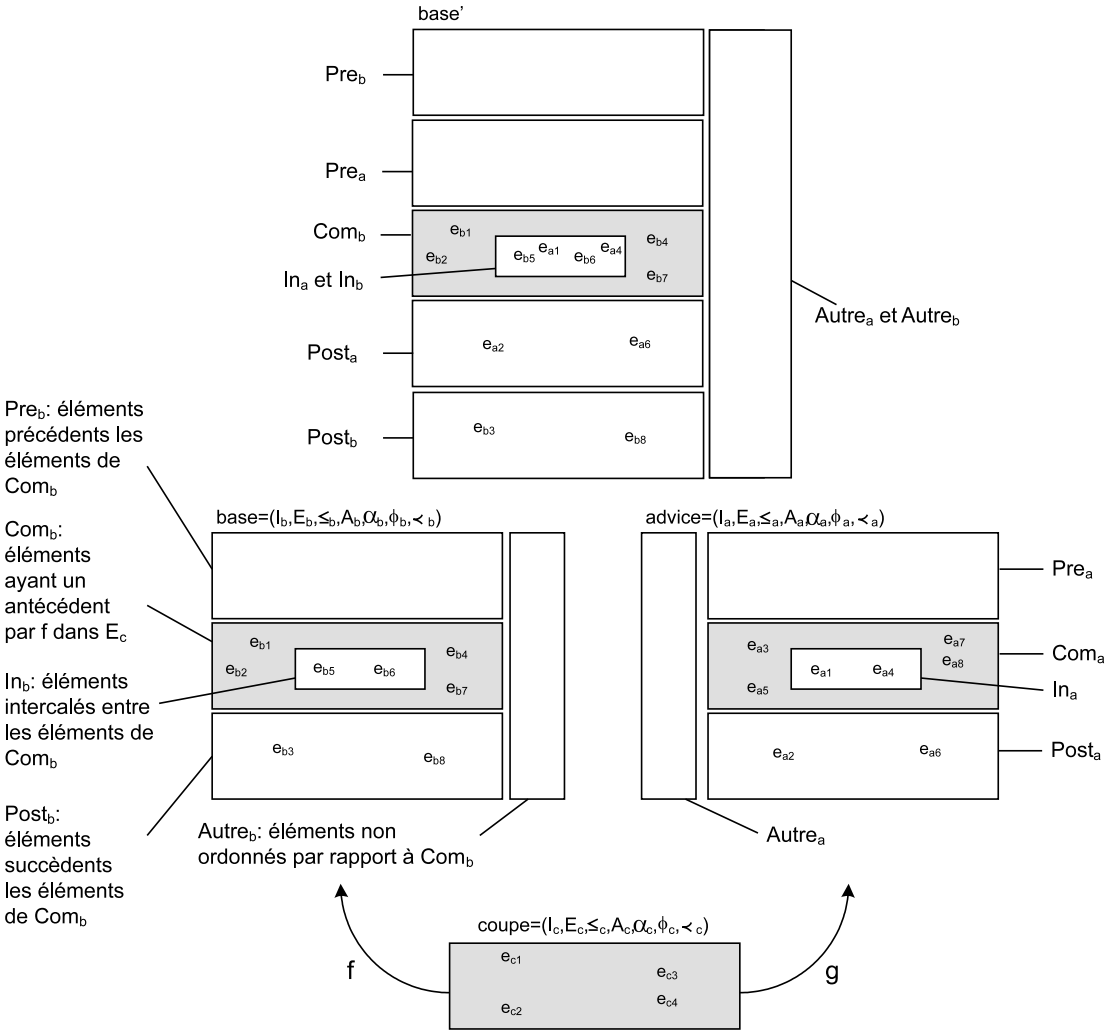


FIG. 4.11 – Vision schématique de l'ordre résultant

$$\forall e \in E, \alpha(e) = \begin{cases} \alpha_1(e) \ \textit{if } e \in E_1 \\ \alpha_2(e) \ \textit{if } e \in E_2 \end{cases} ;$$

$$\forall e \in E, \phi(e) = \begin{cases} \phi_1(e) \ \textit{if } e \in E_1 \\ \phi_2(e) \ \textit{if } e \in E_2 \wedge \nexists i \in I_0 | g_0(i) = \phi_2(e) \\ f_0 \circ g_0^{-1}(\phi_2(e)) \ \textit{if } e \in E_2 \wedge \exists i \in I_0 | g_0(i) = \phi_2(e) \end{cases} ;$$

$$\prec = (\prec_1 \cup \prec_2) \cap E^2$$

La première ligne de la définition de $\leq$ signifie que chaque paire d'événements de E_1 présente dans E (c'est-à-dire les événements E'_1) et ordonnée par $\leq_1$ reste ordonnée par $\leq$. La seconde ligne est équivalente mais pour les événements de E_2. La troisième ligne signifie qu'un événement de E'_1 ayant un élément similaire e'_2 dans E_2, précède un événement e_2 de E_2 présent dans E, si e'_2 précède e_2. La quatrième ligne est équivalente, mais pour la notion de succession. Finalement, la cinquième ligne signifie qu'un événement e_1 de E'_1 qui précéde les "événements communs", précède tous les événements de E_2 localisées sur la même instance (ou l'instance correspondante) de e_1. La dernière ligne est équivalente, mais pour les événements de E'_1 qui succèdent aux "événements communs".

Nous remarquons que les éléments de In et $Autre$ n'apparaissent pas explicitement dans le calcul de l'ordre $\leq$, mais ils sont ordonnés, premièrement, lorsque nous ordonnons les éléments de E'_1 et ensuite par la fermeture transitive. Cependant, certains éléments de In et de $Autre$ ne peuvent pas être ordonnés par rapport à d'éventuels événements de M_2 : nous obtiendrons dans ces cas des corégions (par exemple, sur la figure 4.10, si le message *sauvegarde tentative* était placé sur l'instance *client* entre les messages *log in* et *réessaie*, l'envoi et la réception de ce message ne seraient pas ordonnés par rapport à l'événement e_{a4} correspondant à l'envoi du message *mise à jour*).

Pour conclure cette présentation de la somme amalgamée gauche, nous rappelons les trois points qui la différencient de la somme amalgamée. Dans une somme amalgamée gauche :

- des éléments de l'opérande de gauche peuvent être supprimés ;
- l'ordre résultant sur les événements est différent, car les événements de l'opérande de droite formeront toujours un "bloc" autour duquel les événements de l'opérande de gauche seront ajoutés ;
- les noms des éléments de l'opérande de gauche sont préservés.

4.3.2.3 De la somme amalgamée gauche vers le tissage

Dans le chapitre 1, nous avons défini un aspect comportemental A comme une paire (P, Ad) où P est une expression de coupe, et Ad un advice. L'expression de coupe P permet de détecter des points de jonction dans un bMSC *base*, un point de jonction étant caractérisé par un isomorphisme de bMSC entre P et une partie de *base*. Le chapitre 2 a décrit un processus de détection des points de jonction, et en particulier des algorithmes de détection permettant la construction des isomorphismes entre P et des parties du bMSC *base*. Ces algorithmes de détection fournissent des morphismes injectifs de P vers le bMSC *base*.

Dans ce chapitre, nous avons expliqué que lorsque les points de jonction sont définis comme des motifs, motifs sûrs ou motifs clos, la composition de l'advice avec le scénario de base ne consiste pas simplement à un remplacement des points de jonction par l'advice, car il est nécessaire de tenir compte des messages qui peuvent être intercalés entre les points de jonction ou qui peuvent les entourer. C'est pour cette raison que nous avons proposé la somme amalgamée gauche de deux bMSCs. Or, pour faire la somme de deux bMSCs *base* et *advice*, il est nécessaire de fournir un troisième bMSC (appelé interface) et deux morphismes de bMSCs pour identifier les parties communes et les éléments à supprimer.

Finalement, la combinaison de la phase de détection et de la somme amalgamée gauche, nous permet de tisser un aspect comportemental $A = (P, advice)$ dans un bMSC *base*. La phase de détection nous fournit un morphisme de bMSCs f entre P et *base*. Lors de la phase de composition, nous effectuons la somme amalgamée gauche $base +_f{}_g advice$, où P joue le rôle du troisième bMSC permettant de spécifier les parties communes, où f est obtenu grâce à la phase de détection, et où g doit être spécifié par l'utilisateur. La figure 4.12 résume cette discussion.

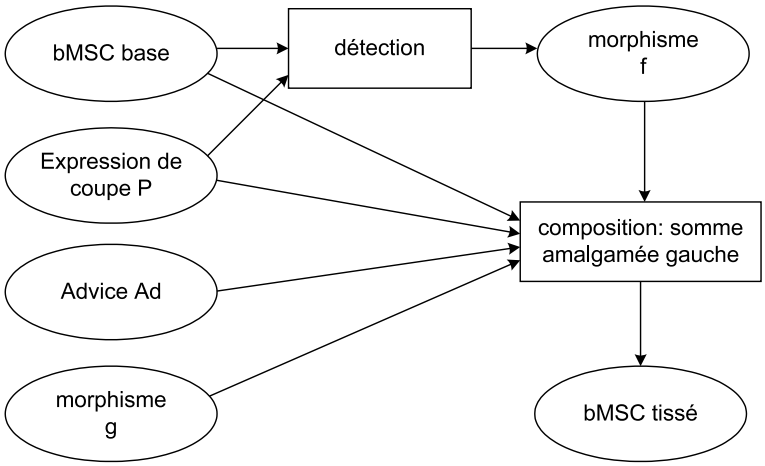


FIG. 4.12 – Processus de tissage dans des comportements finis

Notons que la spécification de g est triviale lorsque l'advice d'un aspect comporte les mêmes noms que ceux de l'expression de coupe, comme pour tous les aspects définis dans le chapitre 2. Mais un intérêt de g est la possibilité de spécifier des advices génériques et réutilisables, comme celui spécifié sur la figure 4.10. En effet, cet advice peut être utilisé lors d'un tissage dans un autre bMSC de base, il suffit seulement d'adapter l'expression de coupe à ce bMSC de base, et de redéfinir g entre l'expression de coupe et l'advice. Notons en toute rigueur qu'à cause de g, la définition d'un aspect comportemental doit être étendue. Un aspect comportemental doit être défini comme un triplet $A = (P, advice, g)$ où g est un morphisme entre P et $advice$, mais nous omettrons de spécifier g lorsque celui-ci est évident.

4.4 Composition par un Produit Fibré

Les sommes amalgamées proposées jusqu'ici permettent de composer des bMSCs. Nous proposons dans cette section un opérateur, appelé *produit fibré*, qui permet de composer des HMSCs.

Cet opérateur n'est pas directement utile dans un processus de tissage d'un aspect comportemental comme défini dans cette thèse. Mais nous avons l'ambition d'utiliser cet opérateur pour tisser des préoccupations de plus haut niveau ou des aspects "structuraux" comme nous le montrons dans la sous-section 4.4.3.

4.4.1 Les HMSCs

Dans le chapitre précédent, nous avons défini un HMSC comme un graphe étiqueté par des bMSCs. Dans ce chapitre, nous proposons une définition équivalente, mais plus adaptée pour définir le produit fibré. Au lieu de directement étiqueter un graphe par des bMSCs, nous étiquetons un système de transitions par des mots, ces mots étant associés à un bMSC. Plus formellement, nous définissons d'abord les systèmes de transitions étiquetées (Labeled Transition Systems, LTS), puis les HMSCs.

Définition 4.6 (Système de Transitions Etiquetées) *Un système de transitions étiquetées (ou LTS) est un n-uplet $\mathcal{S} = (S, \hat{s}, \Sigma, T)$ où : $\mathcal{S}$ est un ensemble de noeuds, $\hat{s}$ est un noeud initial, Σ est un alphabet fini, et $T \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ est un ensemble de transitions. Par la suite, nous noterons par $\alpha(t)$ l'étiquette d'une transition $t = (p, a, q) \in T$. Nous écrirons $p \xrightarrow{a} q$ quand $(p, a, q) \in T$. Le noeud p sera appelé l'origine de t et noté $\lambda_-(t)$. Le noeud q sera appelé la destination de t et noté $\lambda_+(t)$.*

Définition 4.7 (HMSC) *Un HMSC est un n-uplet $H = (\mathcal{S}, \mathcal{M}, \lambda)$ où : $\mathcal{S} = (S, \widehat{s}, \Sigma, T)$ est un LTS appelé automate support de H, $\mathcal{M}$ est un ensemble fini de bMSCs, $\lambda : T \to \mathcal{M}$ associe les transitions aux bMSCs.*

Notons qu'il est évident que cette définition 4.7 est équivalente à la définition 1.13 de HMSC donnée dans le chapitre 1.

4.4.2 Le Produit Fibré

Nous proposons un opérateur fondé sur un produit synchronisé partiel de HMSCs inspiré du produit fibré de systèmes de transitions asynchrones proposé dans [BBC+03]. Le mécanisme d'un produit fibré de deux HMSCs est double. Premièrement, les transitions des deux automates supports sont partiellement synchronisées. Ensuite, les sommes amalgamées des bMSCs attachés aux transitions synchronisées sont calculées pour créer de nouveaux bMSCs. Comme pour la somme amalgamée, la définition formelle du produit fibré de HMSCs est liée à la notion de morphismes :

Définition 4.8 (Morphismes de LTSs) *Soient deux LTSs $\mathcal{S}_1 = (S_1, \widehat{s}_1, \Sigma_1, T_1)$ et $\mathcal{S}_2 = (S_2, \widehat{s}_2, \Sigma_2, T_2)$. Un morphisme de LTSs $f : \mathcal{S}_1 \to \mathcal{S}_2$ est une paire $f =< f_1, f_2 >$*

où $f_1 : S_1 \to S_2$ *est une fonction totale tandis que* $f_2 : T_1 \rightharpoonup T_2$ *est une fonction partielle qui satisfont :*

i) $f_1(\widehat{s}_1) = \widehat{s}_2$;

ii) $t_1 = (p, a, q)$ *dans* T_1 *et* $f_2(t_1)$ *définie implique que* $\exists b \in \Sigma_2, f_2(t_1) = (f_1(p), b, f_1(q))$ *dans* T_2 ;

iii) $t_1 = (p, a, q)$ *dans* T_1 *et* $f_2(t_1)$ *non définie implique que* $f_1(p) = f_1(q)$ *dans* $\mathcal{S}_2$.

La condition *i*) assure que les morphismes préservent les noeuds initiaux. Selon les conditions *ii*) et *iii*), toute transition $t \in T_1$ de $\mathcal{S}_1$ est associée à une transition de $\mathcal{S}_2$ via f_2 si f_2 est définie pour t. En d'autres mots, f_2 définit quelles transitions de $\mathcal{S}_1$ ont des effets observables dans $\mathcal{S}_2$. D'une manière plus pratique, nous pouvons ajouter une transition vide artificielle $p \xrightarrow{\epsilon} p$ à chaque noeud $p \in \mathcal{S}$. Avec cette convention, les transitions de $\mathcal{S}_1$ qui ne sont pas observables dans $\mathcal{S}_2$ sont associées à des transitions vides. Nous pouvons alors réécrire les conditions *ii*) et *iii*) de la définition 4.8 comme suit :

$$t = (p, a, q) \text{ transition de } \mathcal{S}_1 \Longrightarrow \exists a' \in \Sigma_2 \cup \epsilon, f_2(t) = (f_1(p), a', f_1(q)) \text{ transition de } \mathcal{S}_2$$

Cette convention est utilisée par la suite. Par conséquent, il est maintenant supposé qu'un système de transition est complété par une transition vide $p \xrightarrow{\epsilon} p$ sur chacun de ses noeuds.

Le produit synchronisé partiel de LTS a été introduit par Arnold [Arn94]. Il est paramétré par un ensemble de vecteurs de synchronisation, qui est utilisé pour définir les paires d'étiquettes qui doivent être synchronisées. Ici, nous utilisons un ensemble de paires de transitions (au lieu d'un ensemble de paires d'étiquettes) pour cet objectif.

Définition 4.9 (Produit Synchronisé) *Soient deux LTSs* $\mathcal{S}_1 = (S_1, \widehat{s}_1, \Sigma_1, T_1)$ *et* $\mathcal{S}_2 = (S_2, \widehat{s}_2, \Sigma_2, T_2)$. *Une* contrainte de synchronisation C *est un sous-ensemble de* $T_1 \times T_2$. *Les éléments de* C *sont appelés* vecteurs de synchroniation, *et indique les transitions de* $\mathcal{S}_1$ *et* $\mathcal{S}_2$ *à synchroniser. Le* produit synchronisé *de* $\mathcal{S}_1$ *et* $\mathcal{S}_2$ *sous la contrainte de synchronisation* C *est le LTS* $\mathcal{S} = (S, \widehat{s}, \Sigma, T)$ *où :*

- $S = S_1 \times S_2$; $\widehat{s} = (\widehat{s}_1, \widehat{s}_2)$;
- $\Sigma = \big\{(\lambda(t_1), \lambda(t_2)) \mid (t_1, t_2) \in C\big\}$;
- $T = \big\{(s, \sigma, s') \in S \times \Sigma \times S \mid \exists (t_1, t_2) \in C, \sigma = (\lambda(t_1), \lambda(t_2)) \wedge s = (\lambda_-(t_1), \lambda_-(t_2)) \wedge s' = (\lambda_+(t1), \lambda_+(t_2))\big\}$.

La figure 4.13 présente un exemple de produit synchronisé où les transitions b et e sont synchronisés, ainsi que les transitions c et g (les transitions f et d sont synchronisées avec des transitions vides).

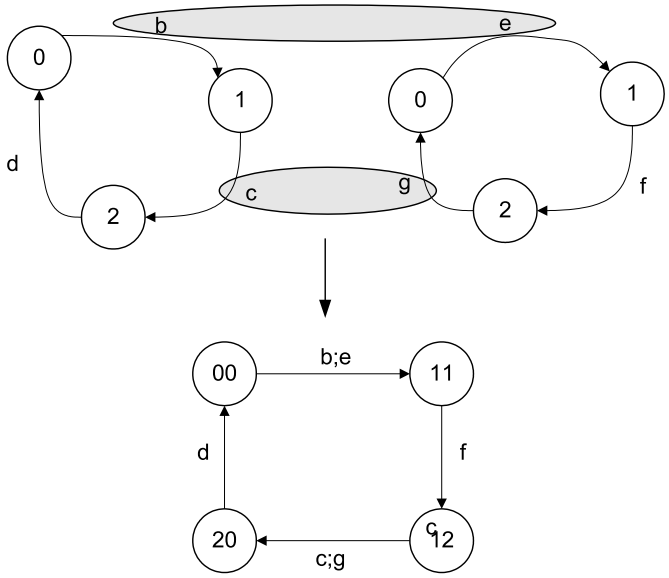


FIG. 4.13 – Un exemple de produit synchronisé

Nous avons déjà défini la notion de morphisme de bMSCs. Un morphisme de HMSCs peut être défini de manière similaire comme une paire (f_1, f_2), où f_1 est un morphisme et f_2 une fonction. Plus précisément, f_1 est un morphisme de LTSs et f_2 est une fonction des morphismes de bMSCs à des transitions.

Définition 4.10 (Morphisme de HMSCs) *Soient deux HMSCs $H_1 = (\mathcal{S}_1, \mathcal{M}_1, \lambda_1)$ et $H_2 = (\mathcal{S}_2, \mathcal{M}_2, \lambda_2)$. Un morphisme de HMSCs $f : H_1 \to H_2$ de H_1 vers H_2 est une paire $f =< f_1, f_2 >$, où $f_1 =< f_{1,1}, f_{1,2} >: \mathcal{S}_1 \to \mathcal{S}_2$ est un morphisme de LTSs, et où f_2 associe les transitions de T_1 aux morphismes de bMSCs de $\lambda_2 \circ f_{1,2}(T)$ vers $\lambda_1(T)$.*

Nous définissons maintenant le produit fibré de HMSCs :

Définition 4.11 (Produit Fibré de HMSCs) *Soient trois HMSCs H_0, H_1 et H_2, et deux morphismes de HMSCs $f =< f_1 =< f_{1,1}, f_{1,2} >, f_2 >: H_1 \to H_0$ et $g =< g_1 =< g_{1,1}, g_{1,2} >, g_2 >: H_2 \to H_0$. Le produit fibré de H_1 et H_2 sous f et g, noté $H_1 {}_f\times_g H_2$, est égal au HMSC $H_1 {}_f\times_g H_2 = (\mathcal{S}, \mathcal{M}, \lambda)$, où :*

- *$\mathcal{S} = (S, \widehat{s}, \Sigma, T)$ est le produit synchronisé de $\mathcal{S}_1$ et $\mathcal{S}_2$ sous $C = \{(t_1, t_2) \in T_1 \times T_2 \mid f_{1,2}(t_1) = g_{1,2}(t_2)\}$ (une transition de H_1 est à synchroniser avec une transition de H_2, si leur image par $f_{1,2}$ et $g_{1,2}$ (resp.) est la même transition dans H_0 ;*
- *$\lambda(t_1, t_2) = \lambda_1(t_1) {}_{f_2(t_1)}+_{g_2(t_2)} \lambda_2(t_2)$ (les bMSCs étiquetants les transitions à synchroniser sont amalgamés) ;*
- *$\mathcal{M} = \lambda(T)$.*

Le HMSC H_0 est appelé interface du produit fibré. Il sert, avec les deux morphismes de HMSCs f et g, à spécifier les transitions à synchroniser. Il fourni également les interfaces des sommes amalgamés à effectuer.

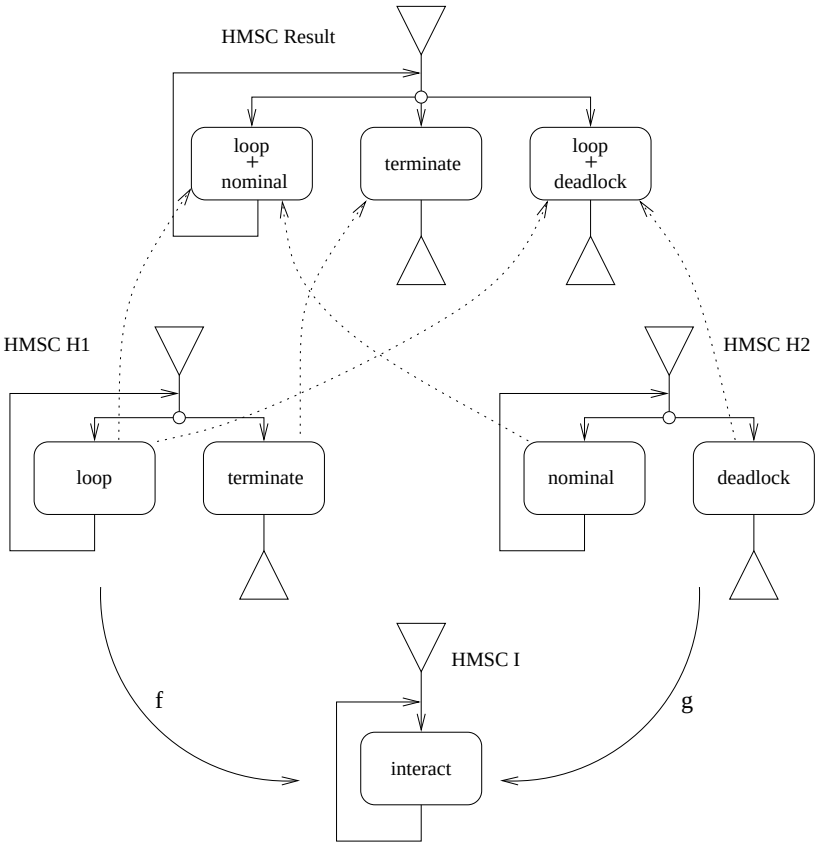


FIG. 4.14 – Produit des HMSCs H1 $_f\times_g$ H2

Illustrons la notion de produit fibré de HMSCs par un exemple. La figure 4.14 montre le produit fibré des HMSC $H1$ et $H2$. Le HMSC $H1$ représente un comportement où le comportement d'un bMSC appelé *loop* peut être effectué un nombre infini de fois avant un bMSC *terminate* qui représente une fin normale du comportement. Le HMSC $H2$ représente un comportement similaire, mais c'est le comportement décrit par le bMSC *nominal* qui est effectué un nombre infini de fois. De plus, au lieu de se terminer normalement, le comportement décrit par $H2$ se termine par un blocage représenté par le bMSC *deadlock*. Lors du produit fibré de $H1$ et $H2$, nous voulons composer les bMSCs *loop* et *nominal*, garder la possibilité de fin qu'est *terminate*, et proposer une deuxième possibilité de terminaison en composant le comportement de *loop* avec *deadlock*.

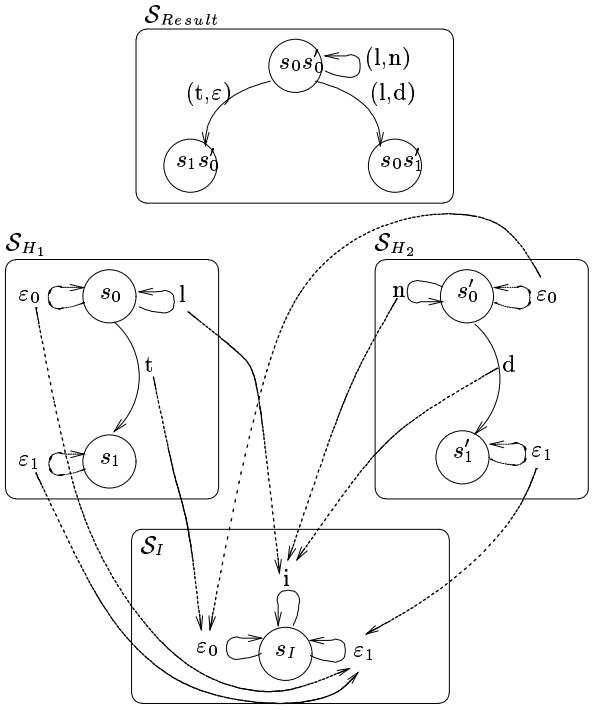


FIG. 4.15 – Automates supports des HMSCs de la fig 4.14

Pour ce faire, la figure 4.14 représente le HMSC I qui est l'interface du produit, ainsi que les deux morphismes de HMSCs $f =< f_1 =< f_{1,1}, f_{1,2} >, f_2 >: H1 \rightarrow H0$ et $g =< g_1 =< g_{1,1}, g_{1,2} >, g_2 >: H2 \rightarrow H0$. Le résultat du produit est représenté par le HMSC *Result*. La figure 4.15 donne les automates supports de $H1$ et $H2$, ainsi que le détail des morphismes $f_{1,2}$ et $g_{1,2}$. La réelle difficulté d'un produit de HMSCs réside dans la définition d'une interface (le HMSC I dans notre exemple), et des morphismes de HMSCs. Comme nous voulons uniquement illustrer le produit fibré de HMSC, nous ne détaillerons pas les fonctions f_2 et g_2 qui fournissent les morphismes de bMSCs nécessaires aux calculs des sommes amalgamée $loop + Nominal$ et $loop + deadlock$. Nous nous focaliserons sur le produit des automates supports.

Détaillons les deux morphismes de LTSs $f_1 =< f_{1,1}, f_{1,2} >$ et $g_1 =< g_{1,1}, g_{1,2} >$ qui sont utilisés pour construire le produit fibré de $H1$ et $H2$. L'automate support S_{H1} de $H1$ (figure 4.15) comporte des transitions étiquetées par l et t, qui sont respectivement associés aux bMSCs *Loop* et *terminate* par λ_1. L'automate support S_{H2} de $H2$ comporte des transitions étiquetées par n et d, qui sont respectivement associés aux bMSCs *nominal* et *deadlock* par λ_2. Par respect de la convention sur les LTSs, des transitions vides ϵ sont ajoutées sur les automates. Les morphismes $f_{1,1}$ et $g_{1,1}$ relatifs aux noeuds des LTSs sont triviaux. Tous les noeuds de S_{H_1} et S_{H_2} sont associés au noeud s_I de S_I. Le morphisme $f_{1,2}$ envoie respectivement les transitions étiquetées par $l, t, \epsilon_0, \epsilon_1$ vers celles étiquetées par $i, \epsilon_0, \epsilon_1, \epsilon_1$. Le morphisme $g_{1,2}$ envoie respectivement les transitions étiquetées par $n, d, \epsilon_0, \epsilon_1$ vers celles étiquetées par $i, i, \epsilon_0, \epsilon_1$. Dans la figure 4.15, les morphismes $f_{1,2}$ et $g_{1,2}$ sont symbolisés par des flèches qui partent des transitions de S_{H_1} (respectivement S_{H_2}) et qui arrivent sur les transitions de S_I.

La contrainte de synchronisation C est construite à partir de f_1 et g_1 : à chaque fois qu'une transition t_1 de S_{H1} a la même image par $f_{1,2}$ qu'une transition t_2 de S_{H2} par $g_{1,2}$, (t_1, t_2) forme un vecteur de synchronisation. Dans notre exemple, C est alors définie par : $C = \left\{ \begin{array}{l} \big((s_0, l, s_0), (s'_0, n, s'_0)\big); \big((s_0, l, s_0), (s'_0, d, s'_1)\big); \\ \big((s_0, t, s_1), (s'_0, \varepsilon_0, s'_0)\big); \big((s_0, \varepsilon_0, s_0), (s'_1, \varepsilon_1, s'_1)\big); \big((s_1, \varepsilon_1, s_1), (s'_1, \varepsilon_1, s'_1)\big) \end{array} \right\}$,

Le produit $\mathcal{S}_{H_1} \times \mathcal{S}_{H_2}$ avec C est représenté par $\mathcal{S}_{Result}$ sur la figure 4.15 (sur ce dernier automate, les transitions vides ne sont pas représentées). Pour obtenir le HMSC *Result*, les bMSCs associés aux transitions de $\mathcal{S}_{Result}$ doivent être amalgamés. Pour une transition donnée (t_1, t_2) de l'automate support du produit fibré, le bMSC $\lambda((t_1, t_2))$ est la somme amalgamée du bMSC associé à la transition t_1 et du bMSC associé à la transition t_2. Les deux bMSCs morphismes neccessaire à la somme amalgamée sont donnés par $f_2(t_1)$ et $g_2(t_2)$.

4.4.3 Produit Fibré et Tissage

Nous rappelons que le produit fibré n'est pas utilisé pour tisser des aspects comportementaux comme définis jusqu'ici. Pour cette raison, nous montrons l'utilité d'un produit fibré dans cette sous-section plutôt que dans le chapitre suivant consacré en partie à l'exposé de cas d'étude relatifs aux aspects comportementaux "traditionnels".

Nous proposons deux types d'applications. Premièrement, nous montrons un exemple de tissage d'aspects comportementaux de haut niveau, qui permettent de modifier l'automate support d'un HMSC. Mais ce type de tissage n'est pas encore complètement défini formellement. Nous l'illustrerons simplement à travers un exemple. Deuxièment, nous proposons un tissage d'aspects plus structurel, qui permet de résoudre des problèmes tels que la non-localité des choix dans un HMSC.

4.4.3.1 Aspects Comportementaux de Haut Niveau

Une des limitations des aspects comportementaux définis dans cette thèse est qu'il n'est pas simple de modifier un comportement d'un point de vue global. Autrement dit, lorsque l'on tisse un aspect comportemental dans un HMSC H, on tisse l'aspect

dans l'ensemble des bMSCs qu'il génère, mais on ne peut pas facilement supprimer ou ajouter des bMSCs dans cet ensemble. Prenons un exemple pour mieux comprendre cette affirmation.

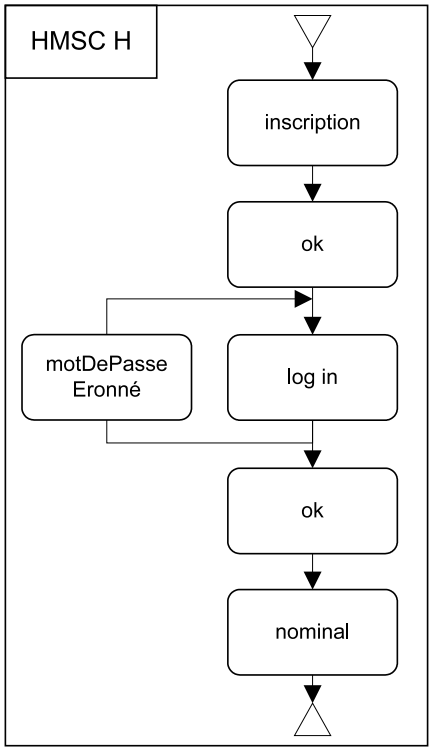


FIG. 4.16 – Scénario de haut-niveau d'une inscription et d'un log in

La figure 4.16 décrit une vue de haut niveau d'un système d'inscription et d'identification (*log in* sur la figure). Après une inscription réussie, il est possible de s'identifier sur un système. Il y a alors deux possibilités : soit l'identification réussit et il est alors possible d'effectuer le comportement *nominal* ; soit l'identification échoue, car le mot de passe n'est pas bon. Dans ce dernier cas, il est possible de retenter de s'identifier. Le problème de ce scénario est qu'il est possible d'essayer un nombre infini de mots de passe puisque le bMSC *motDePasseEronne* est dans une boucle. Ce type de comportement est dangereux du point de vue de la sécurité d'un système, car toute personne malveillante peut tenter d'entrer dans un système en essayant un nombre important de mots de passe.

Avec le type d'aspects comportementaux $A = (P, Ad)$ utilisés jusqu'ici (où P et Ad sont des bMSCs), il n'est pas possible de "casser" le cycle présent dans le HMSC H pour imposer un nombre limité de tentatives, et il n'est également pas possible de créer de nouveaux chemins dans le HMSC H. Cependant, pour faire ce type de modification, un produit fibré de HMSCs est très bien adapté. Considérons l'aspect comportemental de haut niveau décrit par la figure 4.17 où l'expression de coupe et l'advice ne sont pas des bMSCs, mais des HMSCs. Par contre, l'interprétation de l'expression de coupe et

de l'advice peut être la même : à chaque fois que l'on rencontre le comportement décrit par l'expression de coupe H_p, on tisse le comportement décrit par l'advice H_{ad}, où il n'est possible de s'identifier sur le système qu'uniquement à l'aide de deux tentatives.

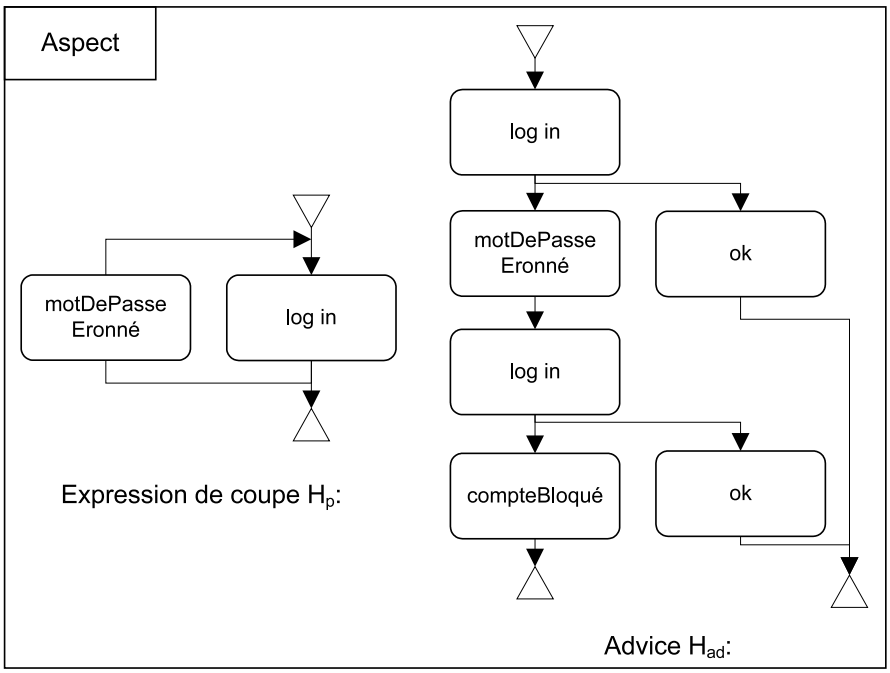


FIG. 4.17 – Aspect comportemental de haut niveau

Le résultat du tissage attendu est représenté dans la figure 4.18 par le HMSC H'. Le cycle présent dans le HMSC H de la figure 4.16 a disparu, remplacé par le comportement de l'advice.

Détaillons maintenant le produit fibré de HMSCs permettant d'obtenir ce résultat. Les HMSCs présentés ne faisant pas apparaître le détail des bMSCs, nous pouvons nous concentrer uniquement sur les automates supports des HMSCs. La figure 4.19 décrit les automates des HMSCs H, H_{ad} et H_p. Le HMSC H_p joue le rôle de l'interface du produit fibré. L'automate $S_{H'}$ est le support du HMSC résultat du produit fibré. La figure 4.20 donne le détail des morphismes de LTSs nécessaires à la construction de la contrainte de synchronisation C, qui est elle-même détaillée plus bas.

$$C = \left\{ \begin{array}{l} \big((s_1, inscription, s_2), (n_1, \epsilon, n_1)\big); \big((s_2, ok, s_3), (n_1, \epsilon, n_1)\big); \\ \big((s_3, \log\text{ in}, s_4), (n_1, \log\text{ in}, n_2)\big); \big((s_3, \log\text{ in}, s_4), (n_3, \log\text{ in}, n_4)\big); \\ \big((s_4, \epsilon, s_4), (n_4, compteBloque, n_5)\big); \\ \big((s_4, motDePasseErrone, s_3), (n_2, motDePasseErrone, n_3)\big); \\ \big((s_4, ok, s_5), (n_2, ok, n_6)\big); \big((s_4, ok, s_5), (n_4, ok, n_6)\big); \\ \big((s_5, nominal, s_6), (n_6, \epsilon, n_6)\big); \end{array} \right\},$$

Nous pouvons supposer que le morphisme de LTSs g_1 est fourni par l'utilisateur. En restant fidèle à l'approche suivie dans cette thèse, le morphisme de LTSs $f_1 : S_H \rightarrow S_{Hp}$ est normalement fourni par une phase de détection. Or, le problème est que, dans

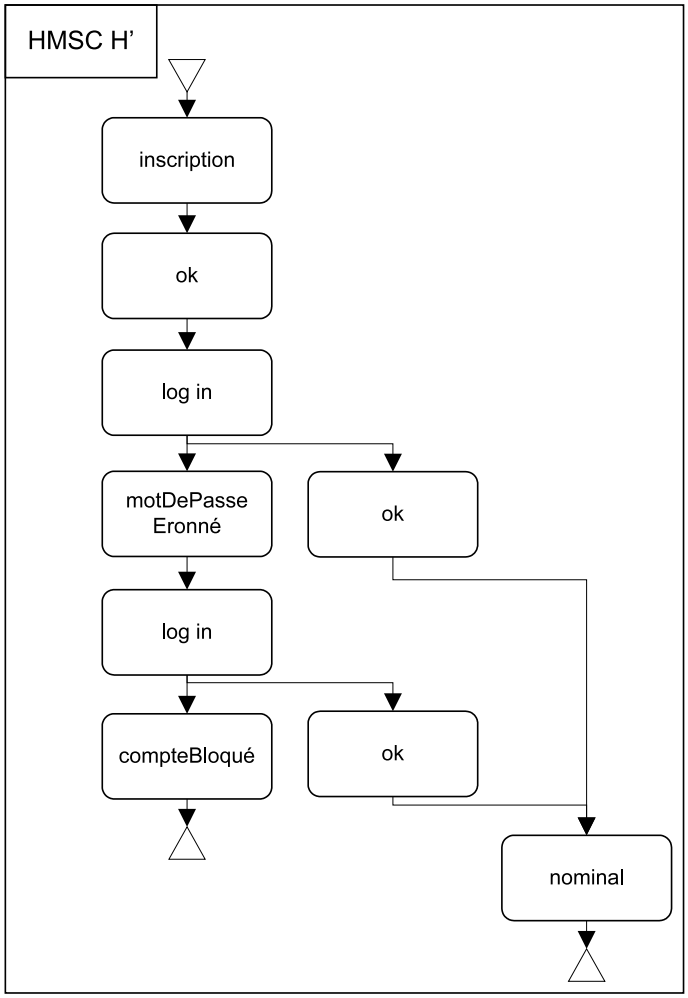


FIG. 4.18 – Résultat attendu du tissage

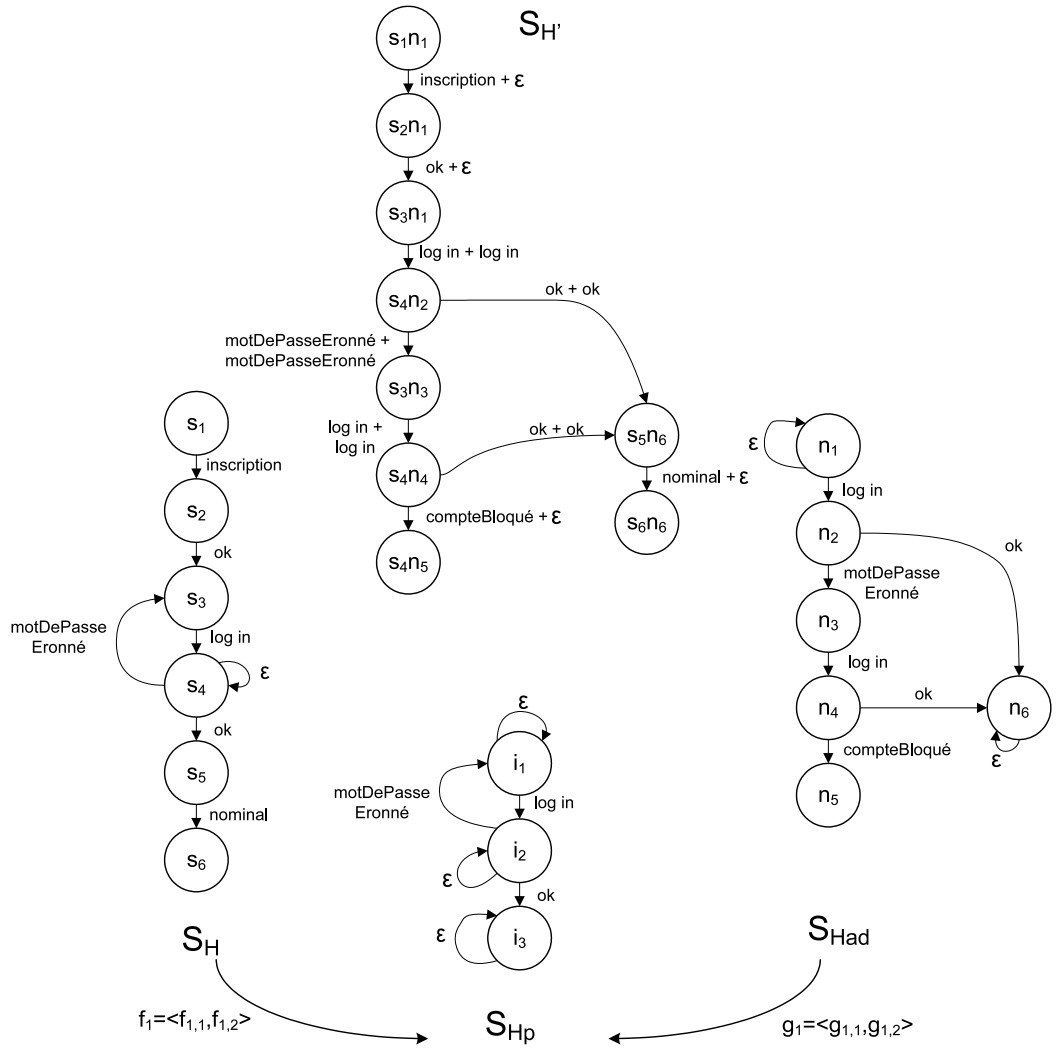


FIG. 4.19 – Produit fibré des automates supports

$f_{1,1}$

noeuds de S_H:	s_1	s_2	s_3	s_4	s_5	s_6
noeuds de S_{Hp}:	i_1	i_1	i_1	i_2	i_3	i_3

$f_{1,2}$

transitions de S_H:	$(s_1$,inscription,$s_2)$	$(s_2$,ok,$s_3)$	$(s_3$, log in, $s_4)$	$(s_4$, ε , $s_4)$	$(s_4$,motDePasseErroné,$s_3)$	$(s_4$,ok,$s_5)$	$(s_5$,nominal,$s_6)$
transitions de S_{Hp}:	$(i_1$, ε , $i_1)$	$(i_1$, ε , $i_1)$	$(i_1$, log in, $i_2)$	$(i_2$, ε , $i_2)$	$(i_2$,motDePasseErroné,$i_1)$	$(i_2$,ok,$i_3)$	$(i_3$, ε , $i_3)$

$g_{1,1}$

noeuds de S_H:	n_1	n_2	n_3	n_4	n_5
noeuds de S_{Hp}:	i_1	i_2	i_1	i_2	i_3

$g_{1,2}$

transitions de S_{Had}:	$(n_1$, ε , $n_1)$	$(n_1$,log in,$n_2)$	$(n_2$,motDePasseErroné,$n_3)$	$(n_3$, log in, $n_4)$	$(n_4$,compteBloqué,$n_5)$	$(n_2$,ok,$n_6)$	$(n_4$,ok,$n_6)$	$(n_6$,ε,$n_6)$
transitions de S_{Hp}:	$(i_1$, ε , $i_1)$	$(i_1$, log in, $i_2)$	$(i_2$,motDePasseErroné,$i_1)$	$(i_1$, log in, $i_2)$	$(i_2$, ε , $i_2)$	$(i_2$,ok,$i_3)$	$(i_2$,ok,$i_3)$	$(i_3$,ε,$i_3)$

FIG. 4.20 – Les morphismes de LTSs $f_1 : S_H \to S_{Hp}$ et $g_1 : S_{Had} \to S_{Hp}$

[MPS98], Muscholl et al. ont montré que l'inclusion d'un comportement d'un HMSC H' dans un HMSC H était indécidable ($\mathcal{L}(H') \subseteq \mathcal{L}(H)$ est indécidable). Pour contourner cette indécidabilité, une première approximation est de ne pas considérer les bMSCs associés aux transitions des automates supports des HMSCs, mais seulement le nom de ces bMSCs. De cette manière, la détection se résume à un problème classique d'inclusion d'un automate dans un autre.

Ce rapide exemple nous a permis de montrer un premier type de tissage qui peut être réalisé en utilisant le produit fibré de HMSCs. Nous qualifions ce tissage, de tissage de haut niveau. Il nous reste cependant à approfondir la phase de détection qui se heurte à des problèmes de décidabilité.

4.4.3.2 Aspects Structuraux

Comme il a été montré dans le chapitre 1, il existe des problèmes dénotant des ambiguïtés dans les HMSCs tels que la non-localité des choix[2]. A travers un exemple, nous allons montrer que l'application d'un produit fibré de HMSCs particulier permet de supprimer la non-localité d'un choix. Même si la résolution de la non-localité des choix ne peut pas être exprimée avec un aspect comportemental classique, cette résolution peut être considérée comme un tissage d'aspect, où la préoccupation transversale (ou l'aspect) est la non-localité des choix. Le processus de tissage reviendrait alors à détecter les choix non locaux (phase de détection), et composer ensuite un comportement rendant les choix locaux (phase de composition). Dans [HJ00, H00], un algorithme de détection de choix non locaux a déjà été présenté. Nous allons maintenant présenter la composition d'un comportement, grâce à un produit fibré, rendant les choix locaux.

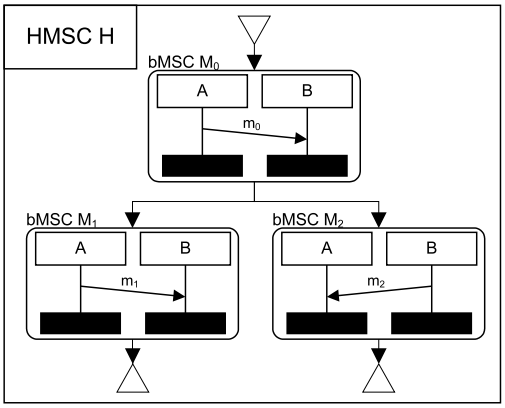


FIG. 4.21 – Un exemple de HMSC non-local

Considérons le HMSC H non-local de la figure 4.21. Ce HMSC contient un noeud non-local, car au niveau de l'alternative, les deux instances A et B peuvent décider du "chemin" à suivre. Le choix n'est pas localisé sur une seule instance. Pour rendre

[2] D'autres de ces problèmes sont présentés dans [H00]

le HMSC local, nous proposons de tisser un protocole de consensus au niveau de l'alternative. Plus précisément, nous voulons tisser un protocole de contrôle qui informera les instances A et B du chemin à prendre. Ce protocole ne devra contenir qu'un seul événement minimum par branche (chemin) de l'alternative, et cet événement minimum devra être effectué par la même instance sur chaque branche, pour assurer la localité du choix. Une première solution est qu'une instance particulière parmi toutes les instances participantes soit désignée comme maître, et qu'elle initie un protocole *token ring* impliquant les autres instances dans un ordre donné. Le *token ring* est alors utilisé pour élire la branche à prendre. Un inconvénient de cette solution est que la symétrie du HMSC est perdue.

Une autre solution est d'ajouter une instance *superviseur* à notre HMSC. Le rôle du superviseur est, premièrement de demander à toutes les instances la branche qu'elles veulent prendre ; deuxièmement de sélectionner une réponse parmi celles reçues ; troisièmement, transmettre ce choix à toutes les instances. Plusieurs politiques de sélection de la réponse peuvent être considérées. Dans cet exemple, nous allons utiliser la politique du *premier arrivé, premier servi*, et nous tisserons la solution avec un superviseur pour transformer le HMSC H. Le HMSC attendu après tissage est représenté sur la figure 4.22, où les bMSCs C_1 et C_2 représentent le protocole avec superviseur à tisser.

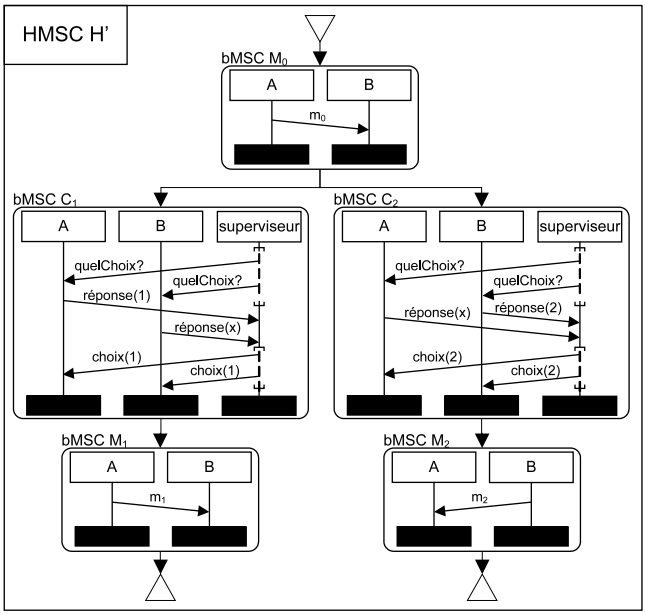


FIG. 4.22 – HMSC local

Pour obtenir le HMSC H' décrit dans la figure 4.22, nous devons insérer les bMSCs C_1 et C_2 entre le noeud de choix et les bMSCs M_1 et M_2 du HMSC de la figure 4.21. Pour ce faire, nous utilisons un produit fibré entre le HMSC H de la figure 4.21 et un HMSC regroupant les bMSCs C_1 et C_2. La figure 4.23 détaille les HMSCs impliqués dans le produit fibré.

Avant d'expliquer l'insertion des bMSCs C_1 et C_2, intéressons nous à l'exemple

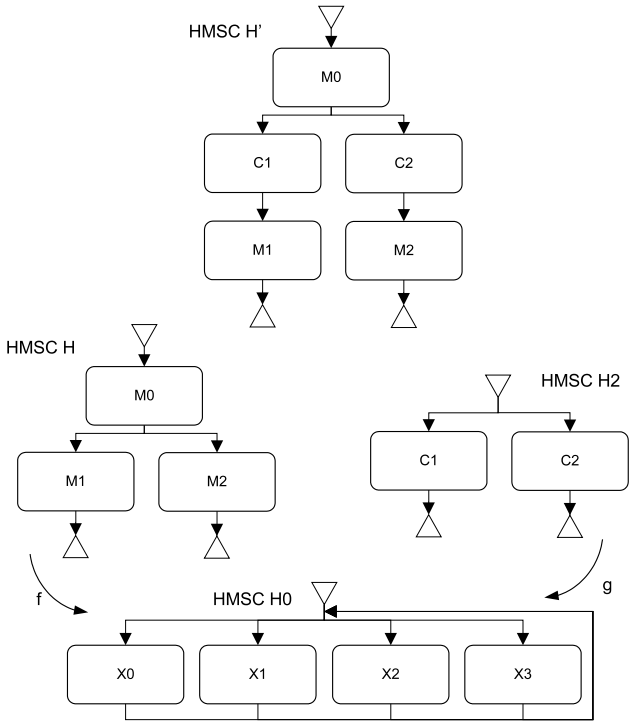


FIG. 4.23 – Produit Fibré de $H \,{}_f\!\times_g H_2$

simple de la figure 4.24, représentant les automates supports de deux HMSCs. Supposons que l'on veuille insérer b avant a. Une solution est de synchroniser b avec la transition étiquetée par ϵ_0, et a avec la transition étiquetée par ϵ_1'. Les morphismes entre les transitions sont détaillés sur la figure 4.24.

Nous pouvons utiliser une technique similaire pour insérer les bMSCs C_1 et C_2 dans le HMSC H non-local de la figure 4.21. Le produit fibré des automates supports est représenté par la figure 4.25. Les morphismes de HMSCs f de H vers H_0 et g de H_2 vers H_0 nécessaire au calcul du produit fibré $H \,{}_f\!\times_g H_2$ sont définis par :

- $f : H = \big(\mathcal{S}_H = (S_H, \hat{s}_H, \Sigma_H, T_H), \mathcal{M}_H, \lambda_H\big) \to H_0 = \big(\mathcal{S}_{H_0} = (S_{H_0}, \hat{s}_{H_0}, \Sigma_{H_0}, T_{H_0}), \mathcal{M}_{H_0}, \lambda_{H_0}\big)$ est une paire $f = \big\langle f_1 =< f_{1,1}, f_{1,2} >, f_2\big\rangle$, où :
  - $f_{1,1} : S_H \longrightarrow S_{H_0}$ associe tous les noeuds de S_H au noeud s_I de S_{H_0}
  - $f_{1,2} : T_H \longrightarrow T_{H_0}$ associe respectivement $(s_0, m_0, s_1), (s_1, \varepsilon_0, s_1), (s_1, m_1, s_2)$ et (s_1, m_2, s_3) à $(s_I, x_0, s_I), (s_I, x_1, s_I), (s_I, x_2, s_I)$ et (s_I, x_3, s_I) ;
  - $f_2 : T_H \to \mathcal{M}_{H_0} \to \mathcal{M}_H$ associe un morphisme de bMSCs nul de $\lambda_{H_0} \circ f_{1,2}(t)$ à $\lambda_H(t)$ à chaque transition $t \in T_H$, c'est-à-dire que si on note μ ce morphisme, nous avons $\mu : M_\varepsilon \to \lambda_H(t)$ (les bMSCs associés aux transitions de H_0 sont des bMSCs vides)
- $g : H_2 = \big(\mathcal{S}_{H_2} = (S_{H_2}, \hat{s}_{H_2}, \Sigma_{H_2}, T_{H_2}), \mathcal{M}_{H_2}, \lambda_{H_2}\big) \to H_0 = \big(\mathcal{S}_{H_0} = (S_{H_0}, \hat{s}_{H_0}, \Sigma_{H_0}, T_{H_0}), \mathcal{M}_{H_0}, \lambda_{H_0}\big)$ est une paire $g = \big\langle g_1 =< g_{1,1}, g_{1,2} >, g_2\big\rangle$, où :
  - $g_{1,1} : S_{H_2} \longrightarrow S_{H_0}$ associe tous les noeuds de S_{H_2} au noeud s_I de S_{H_0}

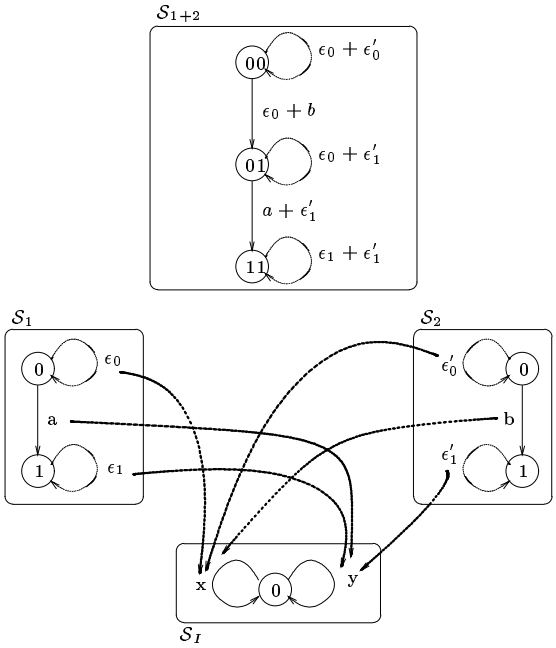


FIG. 4.24 – Une simple insertion

- $g_{1,2} : T_{H_2} \longrightarrow T_{H_0}$ associe respectivement $(s'_0, \varepsilon'_0, s'_0), (s'_0, c_1, s'_1), (s'_0, c_2, s'_2)$, $(s'_1, \varepsilon'_1, s'_1)$ et $(s'_2, \varepsilon'_2, s'_2)$ à $(s_I, x_0, s_I), (s_I, x_1, s_I), (s_I, x_1, s_I), (s_I, x_2, s_I)$ et (s_I, x_3, s_I) ;
- $g_2 : T_{H_2} \to \mathcal{M}_{H_0} \to \mathcal{M}_{H_2}$ associe un morphisme de bMSCs nul de $\lambda_{H_0} \circ g_{1,2}(t)$ à $\lambda_{H_2}(t)$ à chaque transition $t \in T_{H_2}$.

Avec les morphismes f_1 et g_1 définis plus haut, nous pouvons construire la contrainte de synchronisation $C = \big(((s_0, m_0, s_1), (s_I, x_0, s_I)); ((s_1, \varepsilon_0, s_1), (s'_0, c_1, s'_1)); ((s_1, \varepsilon_0, s_1), (s'_0, c_2, s'_2)); ((s_1, m_1, s_2), (s'_1, \varepsilon'_1, s'_1)); ((s_1, m_2, s_3), (s'_2, \varepsilon'_2, s'_2))\big)$. Cette contrainte permet de calculer le produit synchronisé de $\mathcal{S}_H$ et $\mathcal{S}_{H_2}$, dont le résultat est l'automate $\mathcal{S}'_H$ représenté sur la figure 4.25.

Pour obtenir le HMSC final H', il reste à effectuer les sommes amalgamées associées aux transitions de l'automate $\mathcal{S}'_H$. Or toutes les étiquettes "ϵ" sont associées à un bMSC vide (bMSC sans instance et sans événement). De plus, les morphismes f_2 et g_2 ne donnent que des morphismes de bMSCs nuls (le bMSC de départ est vide) pour effectuer les sommes amalgamées. Comme la somme d'un bMSC M avec un bMSC vide, en utilisant des morphismes de bMSCs nuls, donne comme résultat le bMSC M[3], le HMSC H' final est celui représenté par la figure 4.22.

Nous venons de présenter un produit fibré permettant de transformer un HMSC non-local en un HMSC local, en insérant un protocole particulier juste après le noeud de choix non-local. Ce produit fibré peut facilement être automatisé, en particulier, il est relativement facile de créer automatiquement les morphismes nécessaires au produit fibré, car l'opération d'insertion consiste seulement à synchroniser les éléments à insérer

[3]la somme amalgamée $M \,{}_f{+}_g\, M_\epsilon$ avec une interface égal à M_ϵ est égal à M

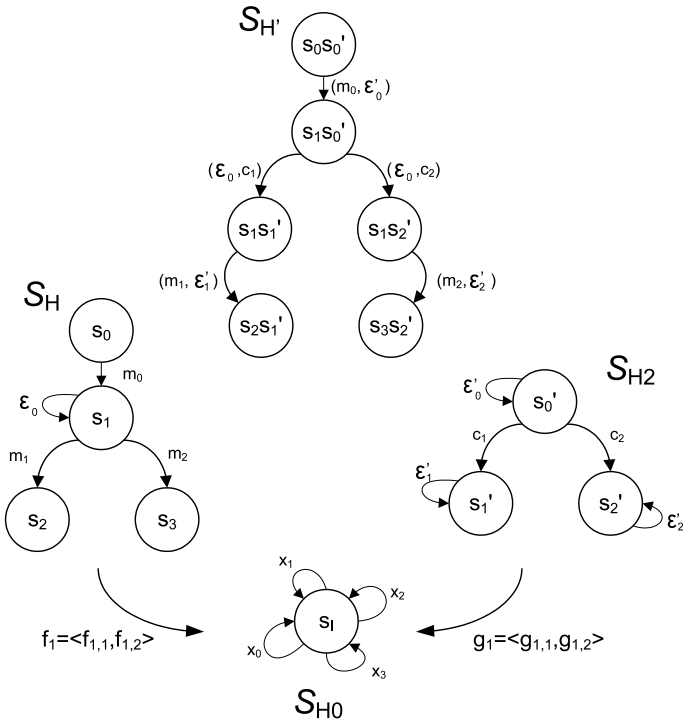


FIG. 4.25 – LTSs des HMSCs à composer

avec des transitions vides ϵ. De plus, notons que le protocole à utiliser pour localiser un choix peut être obtenu automatiquement pour n instances participantes, grâce à l'utilisation de la somme amalgamée, en construisant ce protocole comme une somme de comportements similaires décrivant les échanges entre un superviseur et chaque instance, l'une d'entre elle étant identifiée comme "la première" arrivée. Cet ensemble de comportements peut être obtenu par simple renommage d'un comportement générique en utilisant la somme amalgamée. Les détails de cette création peuvent être trouvés dans [KCH04]. Le produit fibré proposé peut donc être généralisé à des comportements impliquant n instances.

En conclusion, additionné à un algorithme de détection des choix non-locaux, le produit fibré permet d'obtenir un tisseur d'aspects résolvant le problème de la localité de choix. De manière similaire, nous pourrions proposer un tissage permettant de résoudre d'autres problèmes liés aux HMSCs, comme par exemple le problème de la divergence dans les HMSCs [HÓ0].

4.5 Conclusion

Dans ce chapitre, nous avons présenté plusieurs techniques, définies formellement, permettant de composer un comportement de base avec un comportement décrit par un advice. Plus généralement, nous avons proposé des opérateurs de composition de MSCs. La somme amalgamée et la somme amalgamée gauche, nous permettent de composer des bMSCs, la somme gauche étant plus adaptée au tissage d'aspect proprement dit. Le produit fibré permet de composer des HMSCs. Actuellement son utilisation dans un processus de tissage est encore à l'étude, mais les exemples proposés nous donnent de bons espoirs sur son utilité future.

Chapitre 5

Implémentation et Applications

Ce chapitre est divisé en deux parties distinctes. Premièrement, nous décrivons l'implémentation, dans l'environnement Kermeta [MFJ05, Fle06] développé par l'équipe Triskell, du tisseur de scénarios présenté dans cette thèse. Deuxièmement, nous présentons deux applications de ce tisseur de scénarios à travers l'étude d'un système de ventes aux enchères proposé comme étude de cas dans le contexte du réseau européen d'excellence AOSD-Europe. La première application montre un exemple de tissage de plusieurs aspects comportementaux dans un scénario de base conséquent. La deuxième application présente les grandes lignes d'un environnement de test pour modèles développés par aspects.

5.1 Implémentation

Pour exécuter les algorithmes de détection et de composition présentés dans cette thèse, et donc pour fournir un tisseur de scénarios, nous les avons implémentés[1] dans l'environnement Kermeta. Cette section est divisée en deux sous-sections. La première présente l'environnement Kermeta et motive notre choix de cet environnement. La seconde détaille l'implémentation de notre processus de tissage.

5.1.1 Kermeta

Kermeta [MFJ05, Fle06] est une plateforme open-source de métamodélisation développée par l'équipe Triskell. Au coeur de cette plateforme se trouve le langage Kermeta, conçu comme une extension du langage EMOF[2] [OMG04] proposé par l'Object Management Group (OMG). Les langages de métamodélisation tels que EMOF ne permettent de modéliser que des structures (grâce à des concepts tels que les classes, attributs ou associations). Le langage Kermeta ajoute à cela la possibilité de décrire la sémantique et le comportement de ces structures grâce à un langage d'actions. Le langage d'actions

[1] A l'heure actuelle, les algorithmes du chapitre 2 ainsi que la somme amalgamée et la somme amalgamée gauche sont implémentés. Les algorithmes du chapitre 3 sont en cours d'implémentation.

[2] Essential Meta-Object Facilities.

de Kermeta a été conçu spécialement pour la manipulation de modèles. Cela rend le langage Kermeta particulièrement adapté à la définition à la fois de la structure et de la sémantique de métamodèles ou langages, mais aussi à l'écriture de transformations de modèles ou de contraintes sur des modèles [MFV+05].

La structure de Kermeta reste suffisamment proche de EMOF pour être pleinement compatible avec des outils existants tels que le framework de métamodélisation d'Eclipse EMF [BSM+03]. La partie action de Kermeta est constituée d'un langage d'actions impératif qui inclut entre autres :

- des concepts propres au domaine de la manipulation de modèles comme par exemple les associations,
- des concepts orientés-objets classiques tels que l'héritage multiple ou la redéfinition avec une politique de liaison dynamique,
- des expressions et fermetures analogues à celles d'OCL (Object-Constraint Language) [RG02, OCL03]

Une description détaillée de la construction de Kermeta est présentée dans [MFJ05]. L'ensemble de la plateforme Kermeta et des documents sur le langage sont disponibles sur le site de Kermeta (www.kermeta.org).

Kermeta a été utilisé avec succès pour implanter une technique de composition de diagrammes de classes [RFG+05] mais aussi comme langage de transformations de modèles [MFV+05]. L'utilisation de Kermeta pour l'implantation des techniques proposées dans cette thèse a deux intérêts majeurs. Premièrement, cela permet d'encapsuler les algorithmes de détection et de composition directement dans le métamodèle de scénario utilisé. Deuxièmement, du fait de sa compatibilité avec les outils Eclipse, tout modèle manipulé en Kermeta peut facilement être édité, stocké ou visualisé avec un outil Eclipse.

5.1.2 Implémentation du Tissage

Nous rappelons qu'un processus de tissage est décomposé en deux phases : une phase de détection des points de jonction correspondants à une expression de coupe ; et une phase de composition de l'advice au niveau des points de jonction. Le tableau 5.1 récapitule, pour différents types de points de jonction, les détections et les compositions décrites dans cette thèse. Les points de jonction décrits comme des sous-bMSCs ont été définis dans le chapitre 2 et utilisés dans le chapitre 3. Les points de jonction décrits comme des motifs, motifs sûrs et motifs clos ont été définis dans le chapitre 2. Enfin, les points de jonction qualifiés de "sous-HMSC" et de "choix non local" font référence aux exemples présentés dans la section 4.4 du chapitre 4 portant sur le produit fibré.

Dans cette sous-section, nous détaillons uniquement l'implémentation d'un processus de tissage complet, pour des points de jonction définis comme des motifs (motifs, motifs sûrs et motifs clos), dans des comportements finis. L'implémentation des autres types de tissage peut facilement être calquée sur celle présentée.

Comme il a été dit précédemment, le processus de tissage se divise en deux phases : détection et composition. Plus précisément, la phase de détection dans des scénarios finis produit un morphisme entre l'expression de coupe et le modèle de base. La phase

type de points de jonction	processus de détection		processus de composition
	dans des comportements finis	dans des comportements infinis	
sous bMSC	ok	ok (aux limitations près)	remplacement
motif	ok	non	somme amalgamée gauche
motif sûr	ok	non	somme amalgamée gauche
motif clos	ok	non	somme amalgamée gauche
"sous HMSC"	///	non	produit fibré
"choix non local"	///	ok	produit fibré

TAB. 5.1 – Récapitulation des détections et compositions possibles en fonction du type de points de jonction

de composition utilise ce morphisme ainsi que le morphisme faisant le lien entre l'expression de coupe et l'advice pour composer l'advice dans le modèle de base grâce à une somme amalgamée gauche. Chacune de ces phases est une transformation de modèles. La figure 5.1 détaille les modèles d'entrée et de sortie de ces transformations (les ellipses représentent les modèles manipulés, le rectangle noir en haut à gauche de chaque modèle précise à quel métamodèle il est conforme, MSC désignant le métamodèle des MSCs et MMM désignant le métamodèle des morphismes). L'ensemble des modèles manipulés par les transformations sont des MSCs à l'exception des morphismes.

La première étape dans l'implantation de transformations de modèles en Kermeta est la définition des métamodèles d'entrée et sortie des transformations. Les métamodèles ont été définis avec le modeleur Omondo [Omo06] qui offre, en plus de l'édition de modèles UML, la possibilité d'éditer des métamodèles. La figure 5.2 présente le métamodèle de MSCs utilisé, où les opérations des classes ne sont pas spécifiées (ce métamodèle est dans sa majeure partie inspiré du métamodèle de MSCs utilisé dans le logiciel SOFAT développé par L. Hélouët).

Dans ce métamodèle, nous pouvons reconnaître la structure d'automate des HMSCs. En effet, la classe *HMSC* contient une liste de "noeuds" et de "transitions", et une instance de la classe *Transition* est associée à deux instances de la classe *Node* appelées "source" et "target". Nous pouvons aussi constater qu'un bMSC contient une liste d'instances, d'événements et de couple d'événements (la classe *EventCouple*). La classe *EventCouple* permet de spécifier l'ordre partiel sur les événements d'un bMSC. Si une instance de la classe *EventCouple* est associée à deux événements *prec* et *succ*, alors $prec \leq succ$. La classe *Event* est associée à la classe *Instance*, ce qui permet d'associer à un événement l'instance sur laquelle il est localisé. La classe *Event* contient

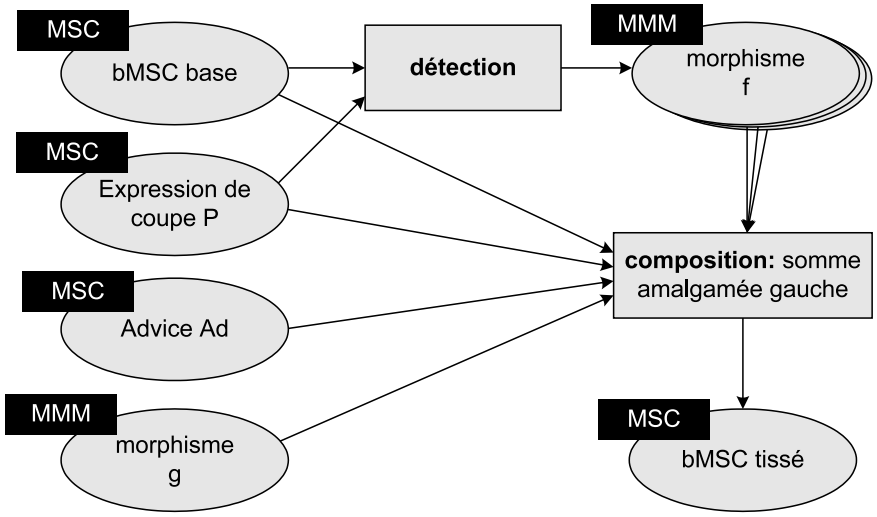


FIG. 5.1 – Transformation de Modèles

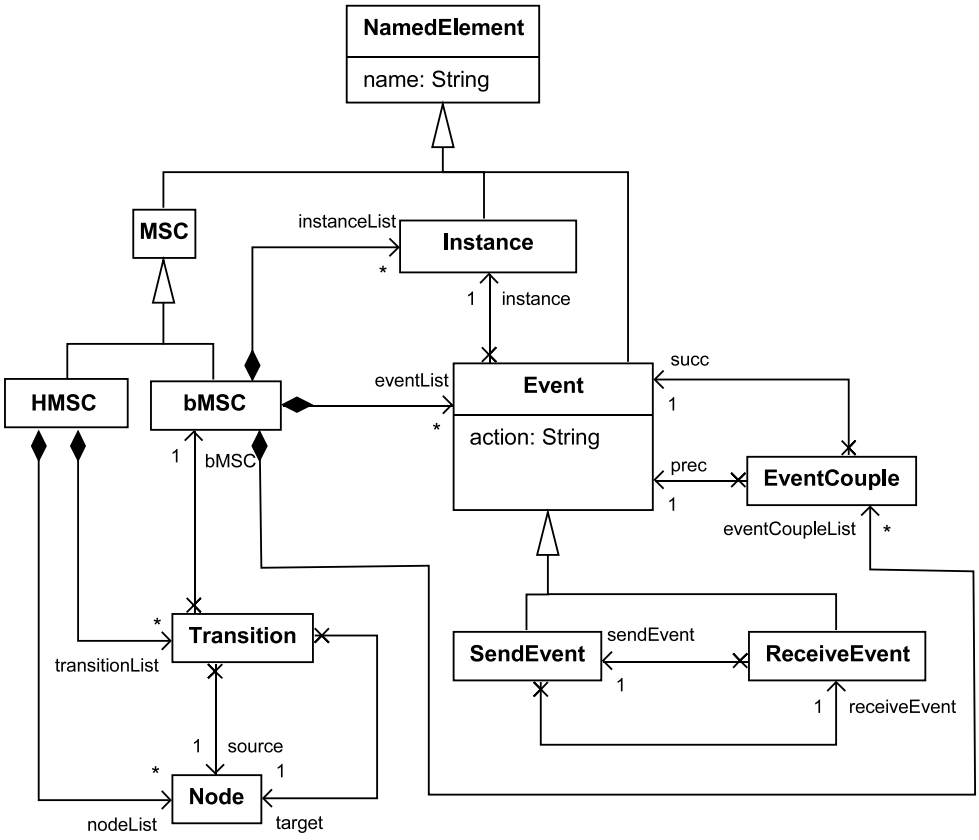


FIG. 5.2 – Un métamodèle de MSCs

un attribut nommé *action*, qui représente le nom du message formé par cet événement. Deux types d'événements sont spécifiés : des événements d'envoi (la classe *SendEvent*) et de réception (la classe *ReceiveEvent*) de messages.

Une fois le métamodèle défini, le framework de métamodélisation d'éclipse (EMF) offre des outils génériques pour créer, éditer et sauvegarder des modèles conformes à ce métamodèle. Kermeta permet d'une part de compléter le métamodèle en spécifiant le corps des opérations contenues dans le métamodèle et d'autre part de manipuler les modèles créés avec EMF. De la même manière, un métamodèle très simple a été défini pour manipuler les morphismes entre diagrammes de séquence. Ce métamodèle est composé d'une classe Morphisme qui encapsule des associations vers deux diagrammes de séquence.

```
package MSC;
using kermeta::standard
using kermeta::persistence
abstract class NamedElement
{
   attribute name : String[1..1]
}
abstract class MSC inherits NamedElement
{ }
class bMSC inherits MSC
{
   attribute events : Event[0..*]
   attribute couples : EventCouple[0..*]
   reference instances : Instance[0..*]

    operation addCouple(e1 : Event, e2: Event): Void is do
         var c:EventCouple
         c:=EventCouple.new
         c.event1:=e1
         c.event2:=e2
         self.couples.add(c)
   end
   operation getMinEvents() : Set<Event> is do
         var preEvents : Set<Event> init Set<Event>.new
         var postEvents : Set<Event> init Set<Event>.new
         self.couples.each{ c |
                  preEvents.add(c.event1)
                  postEvents.add(c.event2)
         }
         result := preEvents.select{ e | e.notIncludedIn(postEvents) }
   end
....
}
```

FIG. 5.3 – Un exemple de code Kermeta

La figure 5.3 montre un exemple de code Kermeta relatif au métamodèle des MSCs. Nous insistons sur le fait que Kermeta possède un langage d'actions permettant de

compléter le corps des méthodes présentes dans les métamodèles. Nous n'avons pas spécifié les méthodes des classes du métamodèle de MSCs de la figure 5.2, mais dans l'exemple de code présenté, nous détaillons deux méthodes de la classe *bMSC*. La première méthode nommée *addCouple* permet d'ajouter une paire ordonnée d'événements $(e1, e2)$, à la liste *couples* des événements ordonnés. La deuxième méthode nommé *getMinEvents* retourne l'ensemble des événements minimaux d'un ensemble d'événements partiellement ordonnés. Dans cette description de méthode, nous pouvons remarquer que Kermeta permet l'utilisation de clôtures lexicales telles que *each* ou *select* qui permettent d'écrire facilement des opérations sur des collections d'éléments.

```
require kermeta   require "../models/MSC.kmt" require "../detectionAlgorithm/Detection.kmt"
require "../amalgamatedSum/LeftSum.kmt"
using kermeta::standard    using MSC

class Weaver {
 operation weave(base : bMSC, pointcut : bMSC, advice : bMSC, g : bMSCMorphism) : bMSC is do

        result := bMSC.new                                                    initialisation
        //Choice of join point policy
        var detection: Detection init ClosedPatternDetection.new
        var sum: LeftSum init LeftSum.new
        var f: bMSCMorphism init bMSCMorphism.new
        var setOfMorphism : Set< bMSCMorphism > init Set< bMSCMorphism >.new

        //Detection Step                                                      détection
        f:= detection.detect(pointcut, base)
        while (f != null)
                setOfMorphism.add(f)
                f:= detection.detect(pointcut, minus(base,f))
        end

        //Composition Step                                                    composition
        setOfMorphism.each{f | result := sum.merge(result, pointcut, advice, f, g)
end
```

FIG. 5.4 – Code Kermeta du tisseur

Une fois les métamodèles définis, les transformations elles-mêmes doivent être conçues et implémentées. Dans notre cas, deux classes Kermeta (*Detection* et *Composition*) ont été définies pour encapsuler les deux phases du tissage. Différentes variantes de l'algorithme de détection ont été implantées par sous classage de la classe *Detection* (une variante pour chaque type de partie d'un bMSC : motif, motif sûr ou motif clos). La somme amalgamée gauche a été implémentée dans la classe *LeftSum* qui sous classe la classe *Composition*. La figure 5.4 montre le code Kermeta de la classe *Weaver* qui tisse un aspect $(pointcut, advice)$[3] dans un bMSC de base nommé *base* (ce sont les paramètres de l'opération *weave*). Un quatrième paramètre est nécessaire pour tisser un aspect : c'est le morphisme de bMSCs g qui associe le bMSC *pointcut* au bMSC *advice*. L'opération *weave* est décomposée en trois parties.

[3] pointcut est le terme anglais d'expression de coupe

La première partie est consacrée à l'initialisation des variables utilisées. La variable *result* représente le retour de l'opération *weave*, c'est donc un bMSC. La variable *detection* représente l'algorithme de détection que nous utiliserons pour détecter les points de jonction. Dans l'exemple proposé, l'algorithme utilisé est celui permettant de détecter des motifs clos (on utilise le constructeur de la classe *ClosedPatternDetection* pour initialiser la variable *detection*). La variable *sum* nous permettra de faire la somme amalgamée gauche de deux bMSCs. La variable f sera le morphisme de bMSCs entre l'expression de coupe et le bMSC de base obtenu grâce à l'algorithme de détection. Enfin, la variable *setOfMorphism* représente un ensemble de morphismes de bMSCs.

La deuxième partie est consacrée à l'obtention du morphisme f entre l'expression de coupe et le bMSC de base obtenu par l'intermédiaire de l'algorithme de détection. Comme il est possible que le bMSC de base contienne plusieurs points de jonction successifs, avant de composer l'advice avec le bMSC au niveau du premier point de jonction trouvé, nous calculons tous les points de jonction corresponds à l'expression de coupe *pointcut*. Pour cela, nous utilisons une boucle, dans laquelle nous calculons un point de jonction dans le bMSC *base* moins le point de jonction découvert précédemment, tant que nous pouvons trouver un point de jonction (tant que le morphisme f n'est pas nul). Chaque morphisme f obtenu est stocké dans l'ensemble de morphismes de bMSCs *setOfMorphism*.

Finalement, la troisième étape consiste à effectuer la somme amalgamé gauche entre le bMSC de base et l'advice au niveau des points de jonction. La somme est donc effectuée pour tous les morphismes de bMSCs f obtenus après la phase de détection.

L'implémentation des algorithmes présentés dans cette thèse, nous a permis d'obtenir un tisseur de scénarios. A l'heure actuelle, les algorithmes du chapitre 2 ainsi que la somme amalgamée et la somme amalgamée gauche sont implémentés. La section suivante présente un exemple de tissage dans des comportements infinis. Cet exemple est pour l'instant accompli "à la main" en attendant que les algorithmes du chapitre 3, portant sur la détection de points de jonction dans des scénarios infinis, soient d'implémentés.

5.2 Applications

Dans cette section, nous présentons tout d'abord un exemple de tissage d'aspects dans des scénarios à travers un exemple relativement important (les scénarios proposés contiennent plus que deux ou trois messages). Ensuite, nous présentons partiellement une application de test de modèles développés par aspects.

5.2.1 Système de ventes aux enchères

Nous présentons un exemple de tissage d'aspects dans des scénarios à travers l'étude d'un système de ventes aux enchères par internet (similaire à eBay). L'objectif de cette présentation n'est pas de proposer une méthodologie de développement logiciel par aspects en utilisant des modèles de scénarios, mais simplement de montrer un exemple,

plus conséquent que ceux déjà proposés, d'un tissage d'aspects comportementaux dans des scénarios.

Les exigences du système à développer sont les suivantes :

Votre équipe à la responsabilité de développer un système de ventes aux enchères en ligne, qui permet à tout utilisateur de vendre ou acheter un bien. La compagnie vous proposant le projet veut rivaliser avec les autres systèmes de ventes aux enchères existants tels que eBay (www.ebay.com) ou uBid (www.ubid.com). Pour cela elle vous demande de réaliser une fonctionnalité supplémentaire qui garantit que toute enchère placée est solvable.

Tous les utilisateurs potentiels du système doivent commencer par s'enregistrer auprès du système. Une fois inscrits, ils doivent s'identifier auprès du système à chaque session. Ensuite, ils peuvent vendre, acheter, ou butiner les enchères actuellement en cours. Les utilisateurs disposent d'un compte dans le système. Ils peuvent alimenter ce compte par des opérations de crédit/débit sur leur carte bancaire. Si un utilisateur veut clore son compte, le système doit transférer la somme restante sur la carte de crédit.

Un vendeur initialise une enchère en informant le système du bien qu'il veut vendre, et en lui associant une enchère de démarrage, la date d'ouverture, la durée des enchères (par exemple 30 jours). Le vendeur peut également indiquer un prix minimum à atteindre (dit de réserve), mais ce n'est pas obligatoire. D'autre part, le vendeur peut annuler la vente tant que les enchères ne sont pas ouvertes.

Pour pouvoir placer une enchère, un utilisateur doit d'abord se joindre à une séance ouverte. Une fois l'utilisateur présent à une séance, il peut faire une enchère. Un utilisateur peut quitter une séance, mais seulement s'il n'a pas fait la plus forte enchère. L'enchère est valide si elle est supérieure à l'enchère courante augmentée d'une somme minimale dont la valeur est variable (par ex. 50 centimes quand l'enchère se situe entre 1 et 10 euros, 1 euro si l'enchère se situe entre 10 et 50 euros, etc.), et si l'acheteur dispose d'un crédit suffisant (compte tenu de toutes ses enchères en cours). Il est possible de renchérir tant que la vente n'est pas close, et on peut participer à autant de séances simultanées qu'on le veut. Quand une séance d'enchères se termine, le système regarde éventuellement si le prix minimum est atteint, puis dépose le montant de la plus haute enchère moins une commission au crédit du vendeur.

A partir de ces exigences, nous pouvons identifier deux types d'utilisateurs potentiels du système. Un vendeur qui met en vente un bien, et un client qui envisage d'acheter un bien. Les différents cas d'utilisation du système relatifs à ces utilisateurs sont facilement identifiables. Pour chaque "action" d'un utilisateur, nous créons un cas d'utilisation. L'ensemble de ces cas est représenté sur la figure 5.5.

A chaque cas d'utilisation, nous pouvons associer un scénario, et en particulier un MSC. La figure 5.6 représente deux HMSCs : les HMSCs *s'incrire* et *joindre une vente*, qui décrivent les scénarios relatifs aux cas d'utilisations du même nom. Ces scénarios représentent un comportement nominal, c'est-à-dire un comportement normal du système (par exemple, quand le client s'inscrit sur le système et que le système accepte cette inscription). Mais ils représentent également les comportements exceptionnels, c'est-à-dire les comportements représentant des situations telles que les situations d'erreurs (par exemple, quand le système refuse l'inscription, car les données nécessaires ne sont

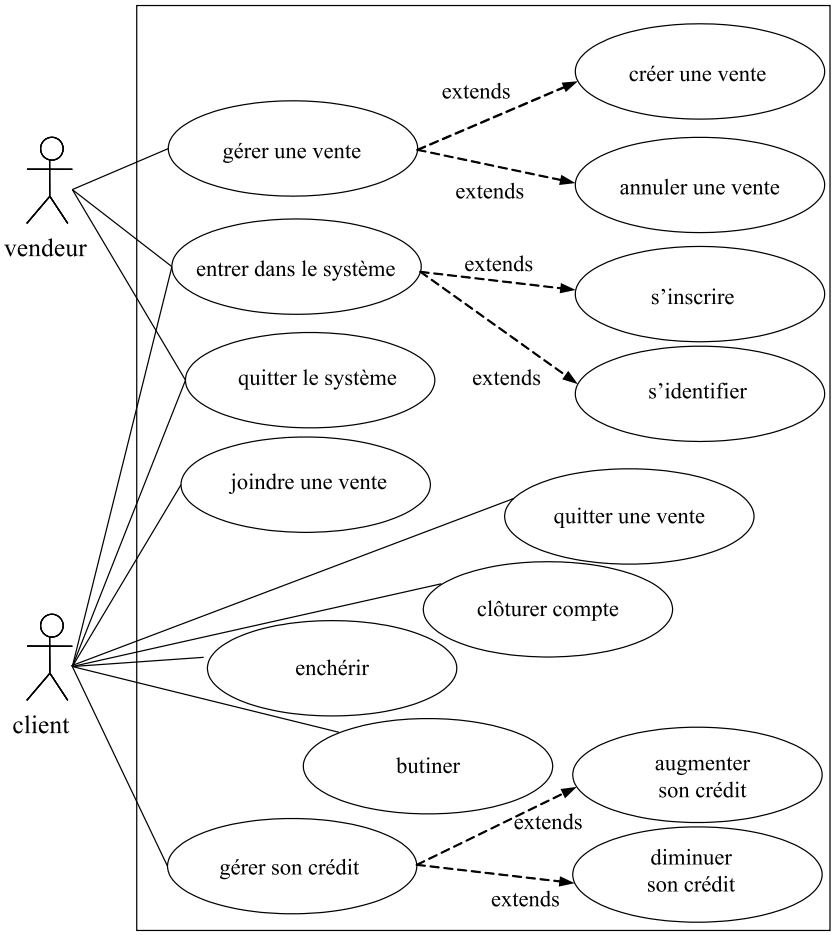


FIG. 5.5 – Cas d'utilisations relatifs au système de ventes aux enchères

pas suffisantes ou pas valides).

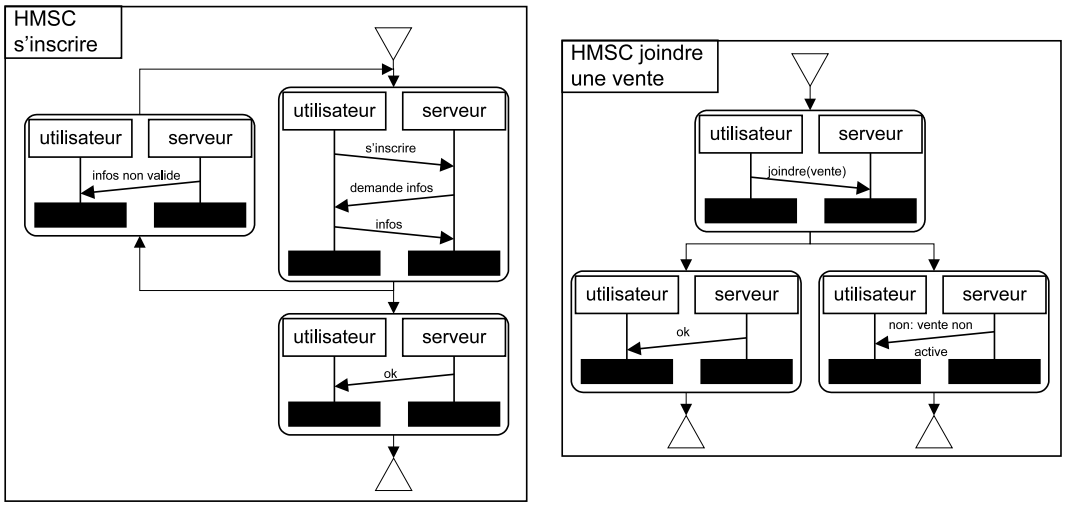


FIG. 5.6 – Exemple de MSCs associés à des cas d'utilisation

Pour obtenir un scénario global du système d'un point de vue "utilisateur", nous ordonnons les différents scénarios relatifs à chaque cas d'utilisation de la figure 5.5. Nous obtenons ainsi le HMSC de la figure 5.7. Pour obtenir un scénario plus détaillé, nous remplaçons chaque transition du HMSC de la figure 5.7 par le scénario détaillé (représentation des interactions entre un utilisateur et le serveur du système) qu'il réfère. Nous obtenons alors le HMSC de la figure 5.8. Sur ce HMSC, par manque de place, nous n'avons pas représenté les scénarios relatifs aux cas d'utilisation *créer une vente*, *annuler une vente*, *clôturer compte* et *diminuer crédit*.

Le scénario de la figure 5.8 est le scénario de base du système. Nous pouvons identifier plusieurs préoccupations transversales, c'est-à-dire plusieurs aspects, relatifs à ce scénario de base. Nous allons détailler quatre de ces aspects, ainsi que leur tissage.

Les aspects sont décrits sur la figure 5.9. Le premier aspect, appelé *aspect sécurité*, permet d'identifier sur le serveur toutes les identifications qui ont échoué. Pour cela, nous utilisons une expression de coupe sur laquelle un message *s'identifier* est suivi par un message autre que *ok* (nous utilisons les caractères génériques, où ! est l'opérateur de négation). L'advice permet d'ajouter un message de sauvegarde lorsqu'un échange de messages recherché est trouvé.

Le deuxième aspect, appelé *aspect persistance*, met en oeuvre une politique de persistance des données. A chaque fois qu'apparaît un échange *utilisateur-serveur*, cet échange est sauvegardé dans une base de données.

Le troisième aspect, appelé aspect *contrôle du temps de réponse*, permet de contrôler le temps de réponse du serveur. Avant chaque échange *utilisateur-serveur*, un timer est déclenché, et après l'échange le timer est remis à zéro.

Finalement, le quatrième aspect, appelé aspect *vérification*, met en oeuvre la vérification d'un montant suffisant sur le compte bancaire d'un utilisateur lorsqu'il en retire

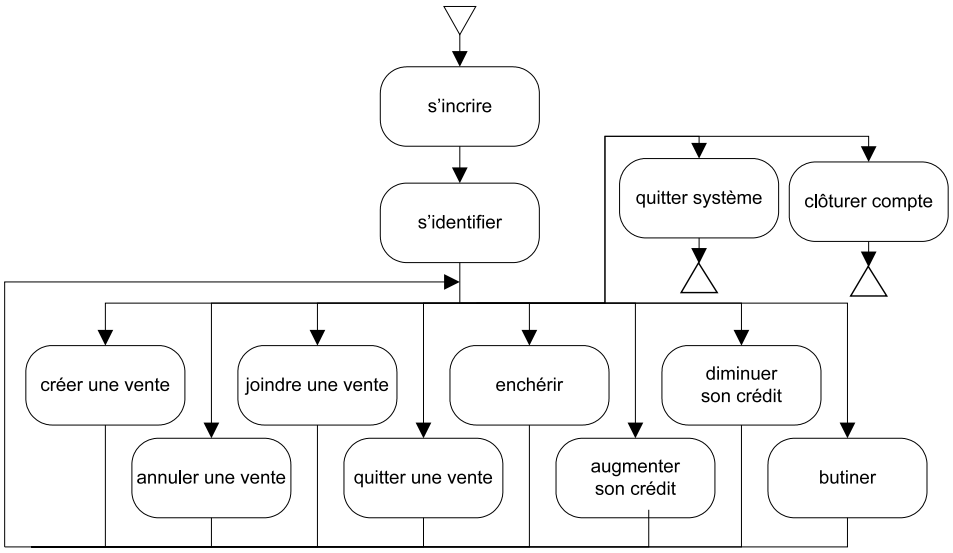


FIG. 5.7 – Scenario Global du système d'un point de vue utilisateur

une certaine somme. Pour ce faire, après chaque demande d'augmentation de la somme que possède un utilisateur sur son compte du système de vente aux enchères, le serveur vérifie la solvabilité du compte bancaire de l'utilisateur.

Ces quatre aspects sont tissés dans le scénario de base de la figure 5.8, en tissant d'abord l'aspect *sécurité*, puis les aspects *persistance* et *contrôle du temps de réponse*, et finalement l'aspect *vérification*. Le résultat de ce tissage est représenté sur la figure 5.10. Notons que puisque le scénario de base définit des comportements infinis, la sémantique des points de jonction est obligatoirement liée à la notion de sous-bMSCs (on détecte uniquement les séquences strictes de messages spécifiées par les expressions de coupe). C'est une restriction imposée par les algorithmes existants. Nous espérons pouvoir lever cette restriction le plus tôt possible.

L'exemple présenté n'est pas une étude exhaustive d'un système de vente aux enchères. Mais le scénario de base utilisé est relativement grand, et il décrit des comportements complexes et infinis. Le tissage de quatre aspects comportementaux a été également présenté. Ce tissage fournit un résultat qui montre l'utilité des techniques proposées. Cependant, les quatre aspects présentés possèdent une expression de coupe fondée sur un échange de messages entre l'utilisateur et le serveur. Comme nous utilisons le tissage dans des comportements infinis, la sémantique des points de jonction est celle des sous-bMSCs. Donc, dès qu'un advice introduit des messages entre un échange utilisateur-serveur, si l'aspect suivant utilise également une expression de coupe fondée sur ce type d'échange, le tissage ne sera plus possible. Dans notre cas, ce n'est que le dernier aspect *vérification crédit* qui insère des messages entre un échange utilisateur-serveur, donc nous ne rencontrons pas de problèmes, mais nous sommes obligés de spécifier un ordre de tissage particulier sur les aspects, et le tissage d'un aspect supplémentaire peut s'avérer problématique.

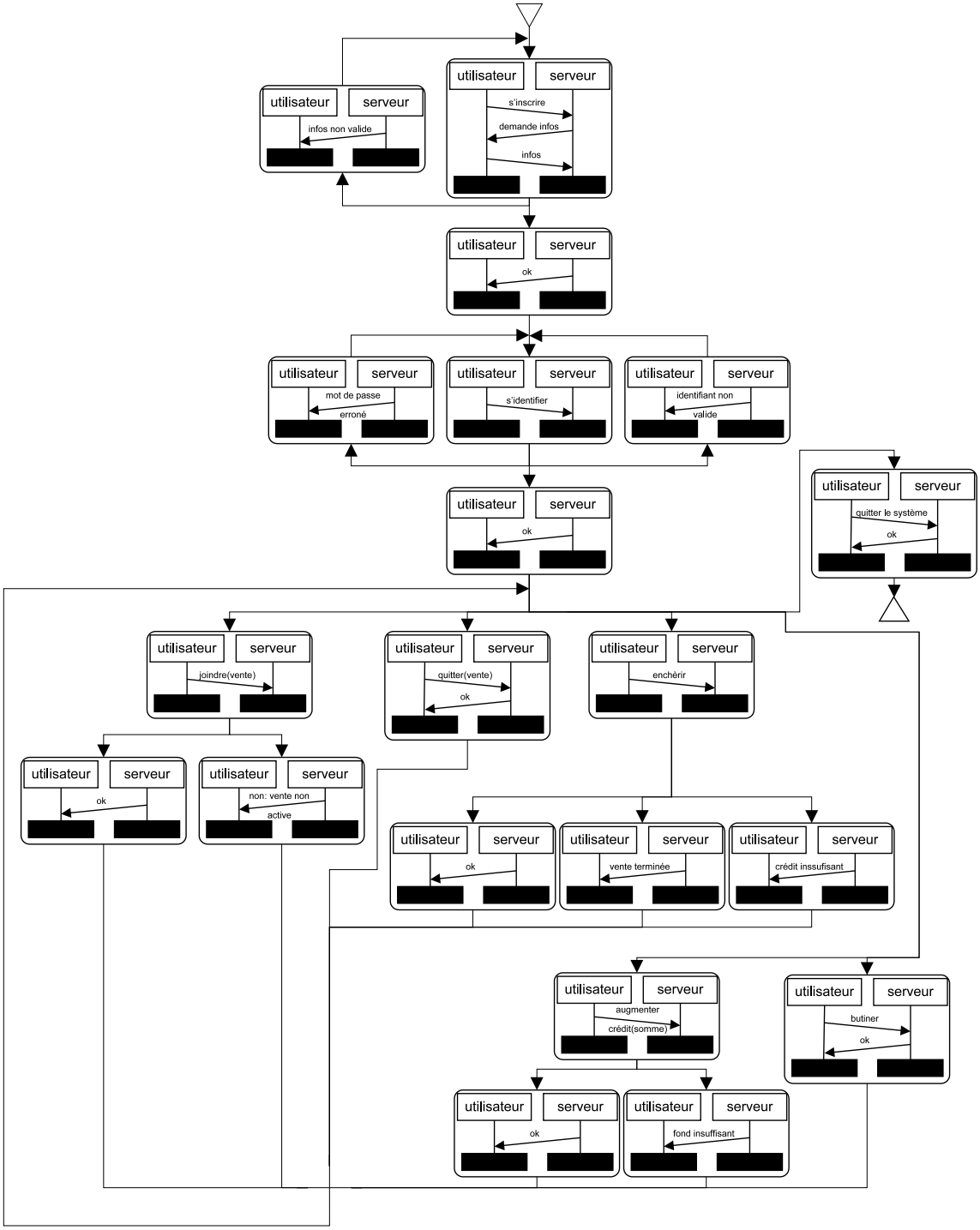


FIG. 5.8 – Scenario détaillé du système

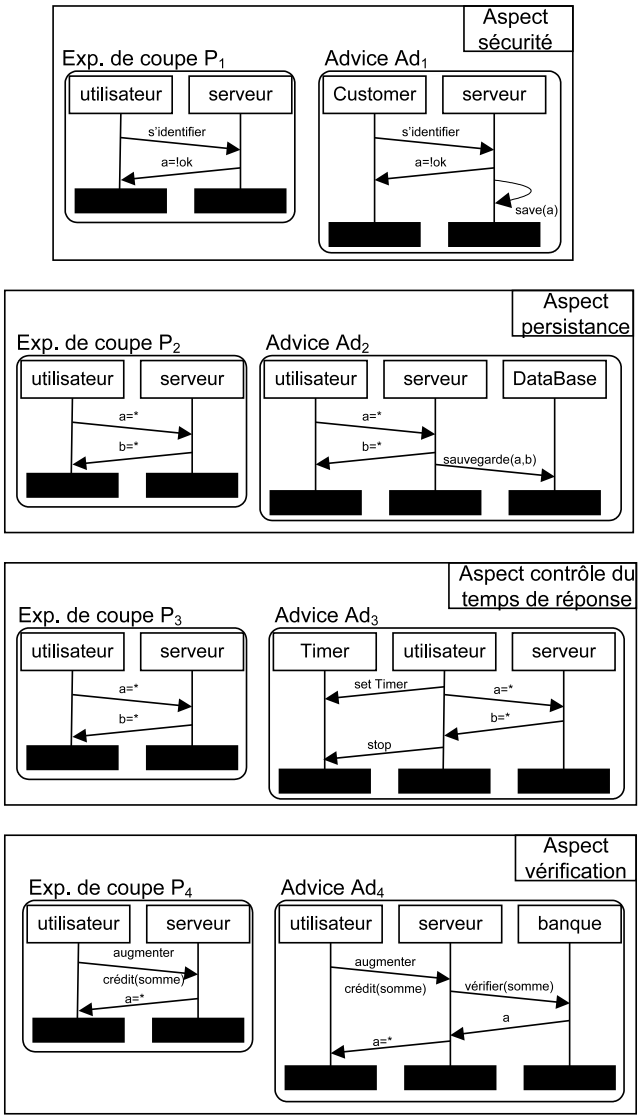


FIG. 5.9 – Les aspects à tisser

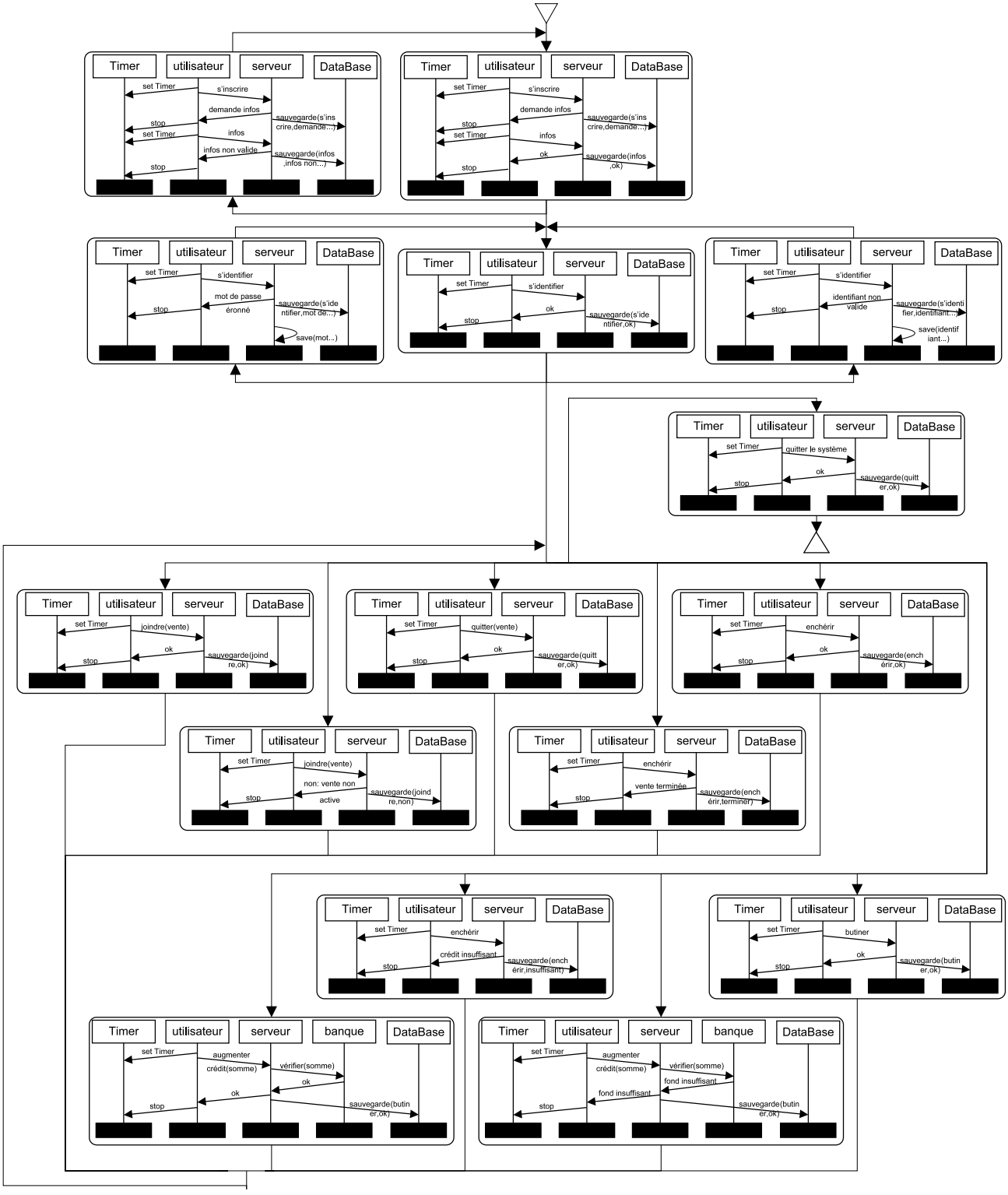


FIG. 5.10 – Résultat du tissage

5.2.2 Application au test de modèles développés par aspects

Dans cette sous-section, nous présentons un environnement de test de modèles développés par aspects, appelé KerTheme. Les travaux portant sur KerTheme [JKBC06b, JKBC06a, JBJC06] étant toujours en court, nous présentons seulement les grandes lignes de cet environnement. Ces travaux sont les fruits d'une collaboration entre l'équipe Triskell et l'équipe Distributed Systems Group (DSG) du Trinity College de Dublin (Irlande). KerTheme est en cours d'implantation dans Kermeta.

5.2.2.1 Présentation Générale

Dans l'Ingénérie Dirigée par les Modèles (IDM), les modèles représentent les vues des systèmes logiciels à différents niveaux d'abstraction, et les transformations de ces modèles spécifient des raffinements d'un de ces niveaux d'abstraction vers un autre [MM03]. Un des bénéfices majeurs de l'IDM est qu'il augmente le niveau d'abstraction dans un développement logiciel [LW05]. Ceci est réalisé à travers une séparation verticale du domaine logique et une mise en oeuvre des technologies dans différents modèles, qui partitionnent les décisions qui doivent être considérées à différentes phases du cycle de développement logiciel. Cette séparation permet de localiser où les décisions sont prises, et par conséquent, où les erreurs peuvent se produire. Une erreur apparaît quand, par exemple, les comportements définis dans les modèles ne sont pas conformes aux exigences. Lorsqu'une erreur se produit dans un modèle, elle est propagée à travers les transformations si elle n'est pas résolue immédiatement. La propagation d'erreur rend difficile la localisation de la source d'erreur et par conséquent rend plus difficile la correction de l'erreur.

La MOA est un domaine particulier de l'IDM qui supporte la décomposition de préoccupations dans des modèles séparés à un même niveau d'abstraction. Les préoccupations sont des "éléments" d'intérêt qui nécessitent d'être traités au cours du développement d'un système logiciel. Cette séparation horizontale étend la séparation verticale de l'IDM en partitionnant davantage les décisions qui doivent être prises à chaque niveau d'abstraction en se basant sur les préoccupations. Un des avantages attendus de la définition des préoccupations dans différents modules est une meilleure maintenabilité des modèles. Les erreurs peuvent être "remontées" à travers différents niveaux d'abstraction puisque les modifications éventuelles sont localisées dans les préoccupations. Pour s'assurer que les erreurs ne sont pas propagées entre les modèles des préoccupations à différents niveaux d'abstraction, une technique pour détecter les erreurs dans les préoccupations modélisées est nécessaire, et plus particulièrement, il est nécessaire de pouvoir tester ces modèles. La décomposition des préoccupations à un niveau de modélisation est facilitée par l'existence de tisseur de préoccupations, pour produire le modèle complet d'un système logiciel. Le tissage est fondé sur des spécifications qui décrivent comment les préoccupations modélisées doivent être composées. Une mauvaise spécification d'un tissage peut introduire des erreurs dans les modèles composés. Pour détecter ces erreurs, il est également nécessaire de tester les modèles composés.

IEEE décrit le test comme : "une activité dans laquelle un système ... est exécuté sous des conditions spécifiées, dont les résultats sont observés ou enregistrés et où une

évaluation est faite de certains aspects du système" [IEE06]. Une interprétation plus abstraite de cette définition est que le test consiste en la validation de la cohérence entre deux vues d'un même système. Pour détecter des erreurs dans des modèles comportementaux, nous avons besoin de considérer deux vues de ces comportements. Pour tester des modèles orientés-aspect, cela signifie que nous avons besoin de deux vues des comportements des préoccupations. Une vue qui décrit précisément comment le comportement des préoccupations doit être exécuté, et une autre qui décrit le comportement attendu de l'exécution. De plus, dans le contexte spécifique de la MOA, certains comportements ne peuvent pas être testés de façon isolée, car ils doivent, au préalable, être tissés avec les autres comportements pour que la phase de test puisse avoir lieu. Dans ce cas, des mécanismes efficaces de traçabilités sont nécessaires pour retrouver la source de l'erreur, soit dans une préoccupation erronée, soit dans spécification de tissage erronée.

Nous proposons un environnement pour MOA, appelé KerTheme, qui offre les caractéristiques requises pour le test, et qui est développé dans la plateforme Kermeta. KerTheme adapte Theme/UML [CB05] qui propose une décomposition symétrique des modèles. KerTheme définit des modules appelés kerThemes. Un kerTheme contient un scénario de haut niveau qui décrit le comportement attendu d'une préoccupation, ainsi qu'un diagramme de classes exécutable qui définit précisément le comportement exécuté. Pour ces deux vues, nous définissons des mécanismes de spécification de composition et des opérateurs de composition. Le processus de test, qui consiste à vérifier la cohérence entre les deux vues, est fondé sur l'analyse des traces d'exécution. Quand nous exécutons le diagramme de classes exécutable, nous sommes capables de construire une trace d'exécution, pour laquelle il est possible de vérifier sa présence dans les comportements définis par le scénario.

5.2.2.2 KerTheme à travers un exemple

Rappel sur Theme

Avant de présenter un exemple de l'approche de test, notons que KerTheme est en partie fondé sur Theme [CB05] qui est une approche de la MOA supportant la décomposition d'un système en des modèles de préoccupations, appelés themes. Les préoccupations sont identifiées en utilisant la méthode Theme/Doc [CB05], et elles sont modélisées en themes en utilisant Theme/UML. Il existe deux types de themes : les themes de base et les themes d'aspect. Un theme et son type sont identifiés à travers l'analyse des exigences et l'identification des comportements qu'ils entrelacent. Un theme d'aspect est utilisé pour concevoir des préoccupations contenant des comportements transversaux. Les autres comportements sont identifiés comme des themes de base.

Un theme contient un diagramme de classe qui décrit la structure de la préoccupation modélisée. Un theme contient également un scénario qui décrit le comportement de la préoccupation modélisée. A la différence d'un theme, dans un kerTheme, les scénarios ne sont pas utilisés pour décrire le comportement du système, mais pour décrire le comportement attendu du système. Les scénarios jouent donc le rôle d'oracle. Dans un kerTheme, le comportement d'une préoccupation est directement décrit par le diagramme de classe exécutable. Comme nous utilisons Kermeta pour implanter

KerTheme, un diagramme de classe exécutable est simplement un programme (ou un modèle) Kermeta.

Présentation de l'exemple et des kerThemes

Pour mieux comprendre KerTheme, nous allons présenter un exemple de système de ventes aux enchères (similaire à celui présenté dans la sous-section précédente). Bien qu'il y ait plusieurs préoccupations d'importances dans un tel système, nous allons uniquement nous intéresser à trois de ces préoccupations : *interface utilisateur*, *inscription* et *persistance*. *Interface utilisateur* est une préoccupation de base représentant le comportement entre un utilisateur et le système à travers une *vue* et un *contrôleur*, lors d'une inscription de l'utilisateur au système. *Inscription* est également une préoccupation de base qui représente le modèle métier d'une inscription d'un utilisateur dans le système. *Persistance* est une préoccupation d'aspect, c'est-à-dire transverse aux préoccupations de base, qui permet de sauvegarder des informations telles que celles fournies lors de l'inscription.

La représentation détaillée de ces préoccupations est décrite par les kerThemes des figures 5.11, 5.12 et 5.13. Chaque kerTheme comporte un diagramme de classe exécutable (même si sur les figures on ne voit que de simples diagrammes de classe) et un scénario.

Les scénarios utilisés dans KerTheme sont des Diagrammes de Séquence (DS) de UML2.0 restreint à des opérateurs de séquence, d'alternative et d'itération. On peut donc considérer que la sémantique des DSs utilisée est exactement la même que celle des MSCs utilisés dans cette thèse (seul l'aspect graphique des DSs utilisés sur les exemples présentés plus loin est différent).

Lorsqu'un kerTheme est une préoccupation de base, le scénario utilisé est un DS habituel, comme sur les figures 5.11 et 5.12. Le scénario de la figure 5.11 décrit les interactions entre l'utilisateur et le système à travers une vue et un contrôleur lors d'une inscription. Une validation des données nécessaire à l'inscription est également effectuée. Cette validation permet, par exemple, de vérifier qu'une adresse électronique est valide. Le scénario de la figure 5.12 décrit l'inscription d'un utilisateur sur le système. Le système (*register*) vérifie que l'utilisateur n'est pas déjà inscrit, et si ce n'est pas le cas, l'utilisateur est ajouté à la liste des inscrits. A la fin du scénario, la vue associée à l'utilisateur est mise à jour.

Lorsqu'un kerTheme est une préoccupation d'aspect, le scénario est en réalité un aspect comportemental, similaire à ceux définis dans cette thèse (composé d'un scénario représentant l'expression de coupe et un scénario représentant l'advice). La figure 5.13 présente l'aspect comportemental lié à la préoccupation *Persistance*. Cet aspect permet de sauvegarder les échanges entre un utilisateur et le système représenté par *register*.

Un kerTheme contient également un diagramme de classe exécutable qui permet d'exécuter les comportements associés aux préoccupations. Les trois figures 5.11, 5.12 et 5.13 présentent une vue des diagrammes de classe exécutable. Sur ces vues, seuls sont représentés les diagrammes de classe, l'executabilité est insufflée à ces classes en ajoutant du comportement dans les méthodes énumérées dans les diagrammes de classe.

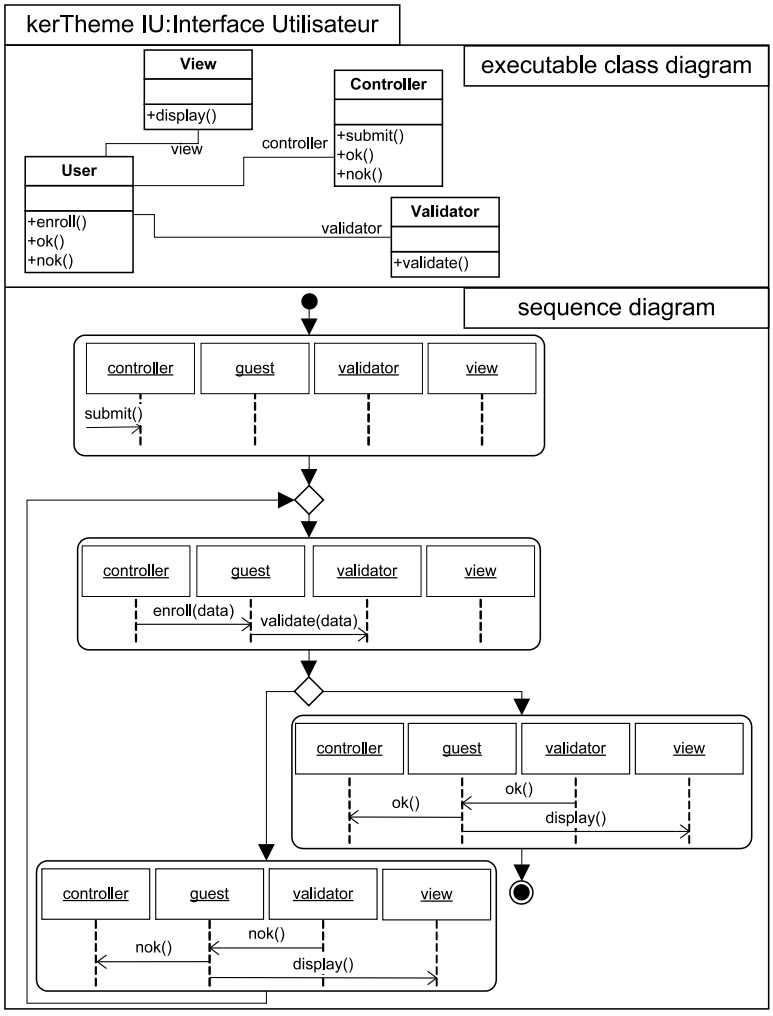


FIG. 5.11 – kerTheme de base Interface Utilisateur

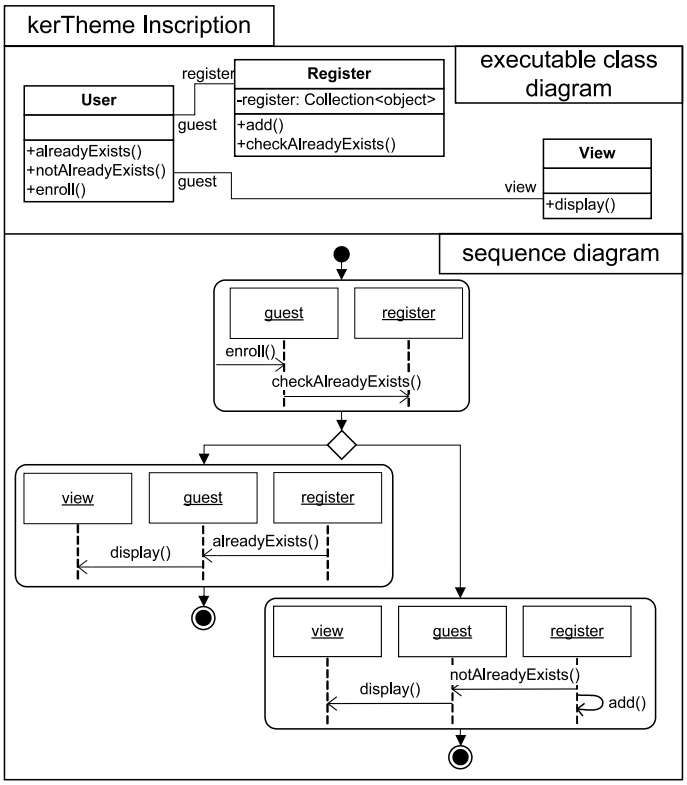


FIG. 5.12 – kerTheme de base Inscription

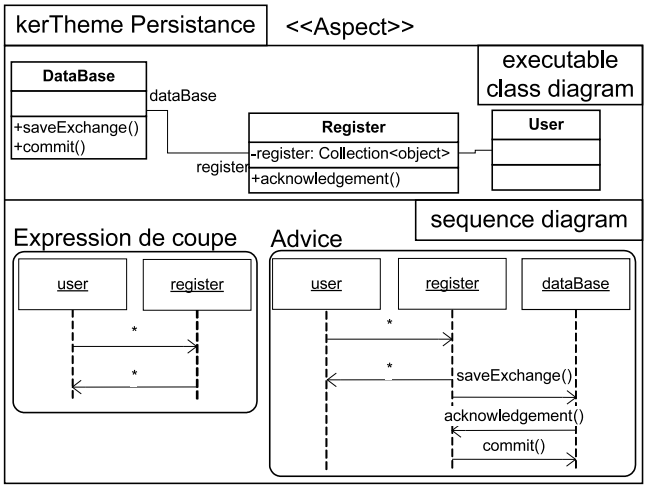


FIG. 5.13 – kerTheme d'aspect Persistance

Pour ajouter ce comportement, nous utilisons Kermeta qui propose le langage d'actions qu'il nous faut. La figure 5.14 représente le code Kermeta relatif aux trois kerThemes étudiés.

Composition des kerThemes

Pour obtenir une description complète d'un système, nous devons être capables de composer les différentes préoccupations, c'est-à-dire les différents kerThemes. Les kerThemes étant à la fois composés de diagrammes de classe exécutables et de scénarios, nous devons être capables de composer ces deux types de modèles. De plus, la composition entre deux kerThemes de base n'étant pas la même qu'entre un kerTheme de base et un kerTheme d'aspect, nous devons au total proposer quatre types de compositions.

La composition de diagrammes de classe est en partie fondée sur l'approche présentée dans le papier [JBJC06], mais cette composition est encore en cours de développement. La composition de deux diagrammes de classe exécutables de kerThemes de base, est essentiellement basée sur la notion de "*fixeur*" permettant de résoudre les problèmes de conflit, comme la présence de deux méthodes de même nom dans les deux diagrammes. La composition de diagrammes de classe exécutables d'un kerTheme de base et d'un kerTheme d'aspect est encore en cours de développement. La composition de ce type de modèle n'étant pas directement liée au sujet de cette thèse, nous ne la détaillerons pas davantage.

La composition de scénarios de deux kerThemes de base est en partie fondée sur la composition présentée dans Theme/UML [CB05] qui consiste principalement à insérer un comportement après ou avant une méthode spécifiée explicitement. Ce type de composition peut être exprimé avec l'opérateur de séquence classique, mais il reste à fournir une expression formelle de cette composition. La composition de scénarios d'un kerTheme de base et d'un kerTheme d'aspect est une application directe du tissage proposé dans cette thèse, puisque le scénario d'un kerTheme d'aspect est un aspect comportemental. La figure 5.15 représente le kerTheme regroupant les trois kerThemes *interface-utilisateur*, *inscription* et *persistance*. On peut remarquer que l'aspect comportemental a été tissé à deux endroits sur la figure 5.15 (aux deux "extrémités" du scénario).

Processus de test

Notre processus de test est basé sur la comparaison de traces. Les scénarios décrivent l'ensemble des comportements attendus. A partir d'une exécution d'un diagramme exécutable d'un kerTheme, nous obtenons une trace d'exécution qui peut facilement être représentée par un scénario fini. Le test consiste alors à vérifier que la trace d'exécution est bien incluse dans le scénario décrit dans le kerTheme. Cela peut se faire grâce aux algorithmes présentés dans [Mus99] (ou en utilisant les algorithmes de détection présentés dans cette thèse).

La phase de test qui vient d'être décrite, peut être appliquée séparément à tous les

Interface Utilisateur

```
class View{
  reference controller : Controller
  operation display(message : String): Void
  is do
     …..
   end
}
class Controller{
  reference user : User
  operation submit(data : Data) : Void is do
       user.enrol(data : Data)
  end
  operation ok() : Void is do
      …..
  end
  operation nok() : Void is do
      …..
  end
}
class User{
  reference validator : Validator
  reference view : View
  reference controller: Controller
  operation enrol(data : Data) : Void is do
   validator.validate(data : Data)
  end
  operation ok() : Void is do
   controller.ok
   view.display(void)
  end
  operation nok() : Void is do
   controller.nok
   view.display(void)
  end
}
class Validator{
  reference user : User
  operation validate(data: Data) : Boolean
  is do
    if data.email.isValid() then
      user.ok
    else
      user.nok
   end
  …..
}
class Data{
  attribute email : Email
}
```

Inscription

```
class User{
  reference register : Register#user
  reference view : View
  operation enrol(data Data) : Void is do
   register.check(data Data)
  end
  operation notAlreadyExists() : Void is do
   register.addToRegister(data)
  end
  operation alreadyExists() : Void is do
   view.display("allready exists")
  end
}
class Register{
  reference user : User#register
  attribute register : Set<Object>
  operation check(data : Data) : Void is do
    if register.exists(data.email) then
      user.alreadyExists()
    else
      user.notAlreadyExists()
    end
  end
  operation addToRegister(data Data) :
  Void is do
    register.add
  end
}
class View{
  operation display(message : String)
  Void is do
    …..
  end
}
class Data{
  attribute email : Email
}
```

Persistance

```
class Database{
  reference register :Register
  operation save() : Void is do
    …..
    register.ack()
  end
  operation commit() : Void is do
    …..
  end
}
class Register {
  reference database :Database
  operation ack() : Void is do
    database.commit
  end
  operation post(data: Data) : Void is do
    database.save(data)
  end
}
class User{}
class Data{}
```

FIG. 5.14 – Code Kermeta des trois kerThemes

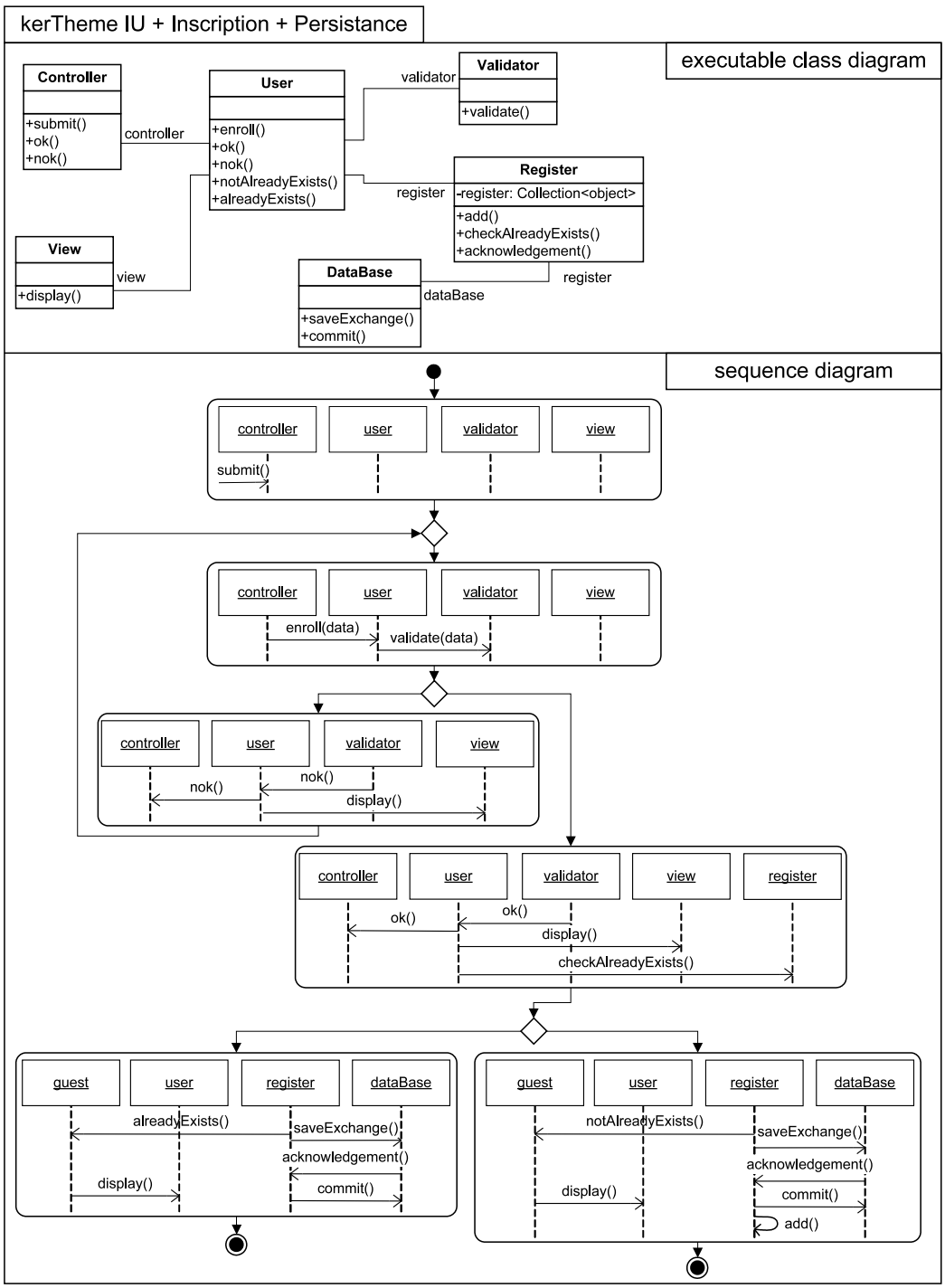


FIG. 5.15 – Résultat de la composition des trois kerThemes

kerThemes de base. Une erreur relative à un kerTheme de base peut donc être facilement localisée. Par contre, un kerTheme d'aspect ne peut pas être testé séparément. En effet, le comportement d'un aspect n'a pas de sens réel tant qu'il n'est pas tissé. Un kerTheme d'aspect ne peut donc être testé qu'une fois tissé.

Pour pouvoir localiser les erreurs dues à un kerTheme d'aspect ou d'une composition, nous utilisons un processus incrémental de test. Dans l'exemple proposé, nous commençons par tester chaque kerTheme de base. Puis nous effectuons la composition des kerThemes de base deux à deux, et entre chaque composition nous testons le résultat. Dans notre exemple, nous testons la composition des kerThemes *interface-utilisateur* et *inscription*. Si une erreur est localisée à ce moment, nous savons que l'erreur provient de la composition des deux kerThemes. Une fois les kerThemes de base composés, nous tissons un à un chaque kerTheme d'aspect (avec un ordre prédéfini), et entre chaque tissage, nous testons le résultat. Une fois encore, le test à chaque tissage permet de localiser facilement une erreur éventuelle.

5.2.2.3 Conclusion

Nous venons de décrire une approche de test pour des modèles orientés-aspects. Le test peut se résumer en une vérification de cohérence entre deux vues comportementales d'un même système. Plusieurs opérateurs de composition et de tissage ont dû être développés pour fournir un environnement de test adapté aux modèles orientés-aspects. Certains de ces opérateurs sont encore en phase de développement. Par contre, nous constatons que le tisseur de scénarios présenté dans cette thèse trouve une application immédiate dans cet environnement de test.

5.3 Conclusion

Les différents algorithmes de détection et de composition présentés dans cette thèse, ont été implémentés[4] avec succès dans la plateforme Kermeta. Les deux cas d'application présentés avaient comme objectif de montrer l'utilité du tisseur de scénarios proposé. Le deuxième cas d'application a notamment montré que le tissage d'aspects à un niveau de modélisation permet de tester relativement tôt (et plus facilement) un système composé de préoccupations de base mais aussi de préoccupations d'aspect.

[4]Les algorithmes du chapitre 3 sont encore en cours d'implémentation

Conclusion et Perspectives

Conclusion :

Le contexte associé aux travaux de cette thèse est celui de la modélisation orientée-aspect (MOA). La MOA est un domaine de recherche récent, et à travers l'état de l'art présenté dans ce document, nous avons constaté que très peu de travaux traitaient du problème du tissage d'aspects dans les modèles dynamiques. Pourtant, à travers plusieurs exemples, nous avons montré que ces types de modèles permettaient d'exprimer simplement des aspects comportementaux bien plus difficiles (voire même impossible) à exprimer à un niveau de programmation avec des langages orientés-aspects.

Nous avons choisi un type de modèle dynamique particulier, celui des Message Sequence Charts (MSCs), qui proposent une sémantique bien définie. En se fondant sur cette sémantique, la contribution principale de cette thèse est la proposition d'un tissage statique d'aspects dynamiques (ou comportementaux).

Notre tissage est qualifié de statique, car pour tisser un aspect, nous proposons une transformation d'un scénario de base exprimé avec des MSCs vers un autre scénario dans lequel l'aspect a été tissé. Un tissage statique a comme avantage majeur de fournir le modèle tissé d'un système, ce modèle pouvant alors être utilisé pour cibler des plateformes non orientées-aspects (comme les systèmes embarqués) ou faire l'objet de validations précoces.

Nous avons défini des aspects comportementaux comme une paire de scénarios finis. L'un pour spécifier un comportement à rechercher, l'autre pour représenter le comportement à composer au niveau des comportements détectés. Pour permettre la mise en oeuvre d'un tissage statique de ce type d'aspects, nous avons proposé des mécanismes de détection et de composition adaptés aux scénarios.

En particulier, dans le chapitre 2 de cette thèse, nous avons élaboré un langage d'expression de coupe qui joue un rôle fondamental pour permettre d'identifier les endroits, appelé points de jonction, où un aspect doit être tissé. Pour cela, nous avons proposé plusieurs sémantiques de points de jonction fondées sur l'ordre partiel induit par les MSCs. Nous avons montré que ces différentes sémantiques favorisent plus ou moins le tissage d'aspects multiples, et nous avons proposé un moyen d'ordonner des points de jonction successifs. Nous avons ensuite proposé un algorithme de détection de ces points jonction dans des comportements finis pour chaque sémantique proposée.

Dans le chapitre 3 de cette thèse, nous avons choisi une sémantique particulière de points de jonction pour fournir un processus de détection dans des comportements

infinis. La contribution majeure de ce chapitre est l'élaboration d'un processus de détection statique de tous les points de jonction présents dans l'ensemble potentiellement infini des comportements qu'un HMSC peut générer. La détection est fondée sur des dépliages de boucles et des permutations permettant l'identification d'un ensemble fini de chemins dans lesquels tous les points de jonction sont présents. Nous avons montré que cette détection n'est pas toujours possible. Nous avons alors donné des conditions suffisantes pour qu'une détection puisse avoir lieu.

Le problème de la composition du comportement représenté par l'advice avec le comportement de base au niveau de chaque point de jonction a été traité dans le chapitre 4. Nous avons proposé des opérateurs de compositions, appelés somme amalgamée gauche, pour des comportements finis, et appelés produit fibré, pour des scénarios infinis. L'intérêt majeur de ces opérateurs est de permettre la spécification de parties communes entre deux opérandes pour produire des résultats qui sont de réelles fusions de comportements ne pouvant pas être exprimés avec des opérateurs de compositions classiques tels que les compositions séquentielles ou alternatives.

Dans le dernier chapitre, nous avons décrit l'implantation de nos processus de détection et de composition dans la plateforme Kermeta. Nous avons finalement présenté des cas d'application montrant l'utilité de notre approche.

Perspectives :

Dans un futur proche, il serait très intéressant de traiter les perspectives propres aux limites de notre approche qui sont énumérées à la fin des chapitres 2, 3 et 4.

- Concernant la détection dans des comportements finis, nous retiendrons deux extensions qui nous semblent intéressantes. Premièrement, il serait utile de proposer des conditions sur les aspects comportementaux, permettant de dire, par exemple, si deux aspects commutent, c'est-à-dire s'ils peuvent être tissés dans n'importe quel ordre, ou s'ils sont en conflit. Dans ce dernier cas, il est nécessaire de spécifier un ordre de tissage de ces aspects, ou bien de modifier ces aspects pour résoudre le conflit. Deuxièmement, il nous semble intéressant d'étendre le mécanisme de caractères génériques proposés pour permettre d'utiliser ces caractères sur les noms des instances en plus de leurs utilisations sur les noms des messages.
- Concernant la détection dans des comportements infinis, la définition de conditions nécessaires (en plus des conditions suffisantes proposées dans ce document) permettrait d'identifier des classes du problème de détection pour lesquelles les algorithmes terminent toujours. De plus, une autre amélioration possible est l'utilisation d'autres sémantiques de points de jonction, similaire à celles utilisées dans les comportements finis.
- Concernant le processus de composition, nous n'avons pas encore totalement valorisé le produit fibré dans le cadre de tissage de scénarios. Un objectif est maintenant de proposer un mécanisme de détection permettant de compléter le mécanisme de composition fourni par le produit fibré, pour ainsi obtenir un tisseur de scénarios de plus haut niveau d'abstraction.

Dans une perspective plus large, nous travaux se placent dans le cadre du tissage et de la composition de modèles. Or, durant le cycle de développement de logiciel, il est le plus souvent utilisé plusieurs types de modèles : diagrammes de cas d'utilisation, diagrammes d'activité, machines à états, scénarios, etc. Dans ce contexte, nous identifions trois perspectives majeures :

- Premièrement, pour ne pas restreindre la modélisation orientée-aspect à l'utilisation d'un tisseur pour un modèle particulier comme les scénarios, il est nécessaire de proposer des tisseurs pour d'autres types de modèles. Pour ce faire, le produit fibré proposé dans cette thèse peut être un outil intéressant pour composer des modèles à structure d'automate.
- Deuxièmement, si plusieurs sortes de modèles sont utilisées, lorsque l'on effectue le tissage d'un aspect avec un modèle particulier (par exemple avec des scénarios), pour que les autres modèles (par exemple les diagrammes de classe) restent cohérents avec la spécification du modèle tissé, il est nécessaire de "mettre à jour" les autres modèles. Un moyen automatique de mise à jour est alors à envisager.
- Troisièmement, pour atteindre les objectifs annoncés de l'utilisation d'approches de modélisation orientées-aspect comme la validation précoce, le ciblage de plateformes non orientées-aspect, ou encore une amélioration du contrôle des variations et l'évolution de logiciels (dans le contexte des lignes de produits par exemple), il est nécessaire de proposer des méthodologies d'utilisation de tisseurs de modèles.

Bibliographie

[ABV92] Mehmet Aksit, Lodewijk Bergmans, and Sinan Vural. An object-oriented language-database integration model : The composition-filters approach. In *ECOOP '92 : Proceedings of the European Conference on Object-Oriented Programming*, pages 372–395. Springer-Verlag, 1992.

[AEB01] Omar Aldawud, Tzilla Elrad, and Atef Bader. A uml profile for aspect-oriented modeling. In *Workshop on Advanced Separation of Concerns in Object-Oriented Systems (OOPSLA'01)*, Tampa, Florida, October 2001.

[AEB03] Omar Aldawud, Tzilla Elrad, and Atef Bader. Uml profile for aspect-oriented software development. In *3rd International Workshop on Aspect Oriented Modeling (In conjonction of AOSD'03)*, Boston, Massachusetts, March 2003.

[AEM00] R. Alur, K. Etessami, and Yannakakis M. Inference of Message Sequence Charts. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE)*, Limerick, Ireland, Juin 2000.

[AHP96] R. Alur, G. Holzmann, and D. Peled. An analyser for mesage sequence charts. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 35–48. Springer-Verlag, 1996.

[Arn94] André Arnold. *Finite transition systems.* Prentice-Hall. Prentice-Hall, 1994.

[AWK04] J. Araujo, J Whittle, and Kim. Modeling and composing scenario-based requirements with aspects. In *Proceedings of RE 2004*, Kyoto, Japan, September 2004.

[BAL97] H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In E. Brinksma, editor, *Proceedings of the Third International Workshop on Tools and Algorithms for the Construction and Analysis of Systems TACAS'97*, volume 1217 of *Lecture Notes in Computer Science*, pages 259 – 274, Enschede,The Netherlands, April 1997. Springer-Verlag.

[Bar05] Olivier Barais. *Construire et Maîtriser l'Evolution d'une Architecture Logicielle à base de Composants.* PhD thesis, Université des Sciences et Technologies de Lille, 2005.

[BBC+03] Marek A. Bednarczyk, Luca Bernardinello, Benoît Caillaud, Wieslaw Pawlowski, and Lucia Pomello. Modular system development with pullbacks.

In *Applications and Theory of Petri Nets 2003, 24th International Conference, ICATPN, Eindhoven, The Netherlands, June 23-27,*, Lecture Notes in Computer Science, pages 140–160. Springer, 2003.

[BC04] Elisa Baniassad and Siobhan Clarke. Theme : An approach for aspect-oriented analysis and design. In *ICSE '04 : Proceedings of the 26th International Conference on Software Engineering*, pages 158–167. IEEE Computer Society, 2004.

[BLMD06] O. Barais, J. Lawall, A-F. Le Meur, and L. Duchien. Safe integration of concerns in a software architecture. In *13th Annual IEEE International Conference on the Engineering of Computer Based Systems (ECBS)*, Potsdam, Germany, March 27th-30th, 2006.

[BSM+03] Frank Budinsky, David Steinberg, Ed Merks, Ray Ellersick, and Timothy Grose. *Eclipse Modeling Framework*. The Eclipse Series. Addison Wesley Professional, 2003.

[CB05] Siobhán Clarke and Elisa Baniassad. *Aspect-Oriented Analysis and Design : The Theme Approach*. Number ISBN : 0-321-24674-8. Addison Wesley, 2005.

[CCMV05] Olivier Caron, Bernard Carré, Alexis Muller, and Gilles Vanwormhoudt. Mise en oeuvre d'aspects fonctionnels réutilisables par adaptation. *Numéro spécial de la revue L'OBJET : Programmation par Aspects*, volume 11-3, 2005.

[Chi05] Rashid A. Sawyer P. Bakker J. Pinto Alarcon M. Garcia A. Tekinerdogan B. Clarke S. Jackson A. Chitchyan, R. Survey of aspect-oriented analysis and design. Technical report, AOSD-Europe Project Deliverable No : AOSD-Europe-ULANC-9. Editor(s) : R. Chitchyan, A. Rashid., 2005.

[Cla01] Siobhan Clarke. *Composition of Object-Oriented Software Design Models*. PhD thesis, Dublin City University, 2001.

[CM98] C. Consel and R. Marlet. Architecturing software using a methodology for language development. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *Proceedings of the 10th International Symposium on Programming Language Implementation and Logic Programming*, volume 1490 of *LNCS*, pages 170–194, Pisa, Italy, September 1998.

[CW01] Siobhan Clarke and Robert J. Walker. Composition patterns : An approach to designing reusable aspects. In *Proceedings of ICSE'01*, pages 5–14, 2001.

[CW02] Siobhan Clarke and Robert J. Walker. Towards a standard design language for aosd. In *First International Conference on Aspect-Oriented Software Development (AOSD)*, Enschede, The Netherlands, April 2002.

[CW04] Siobhan Clarke and Robert J. Walker. Generic aspect-oriented design with theme/uml. *Aspect-Oriented Software Development*, Addison-Wesley, 2004.

[DFS02] Rémi Douence, Pascal Fradet, and Mario Südholt. A framework for the detection and resolution of aspect interactions. In *Proceedings of GPCE'02*, LNCS. Springer, 2002.

[Dij76] Edsger Wybe Dijkstra. *A discipline of programming.* Prentice-Hall Series in Automatic Computation, Englewood Cliffs : Prentice-Hall, 1976, 1976.

[DMS01] Rémi Douence, Olivier Motelet, and Mario Südholt. A formal definition of crosscuts. In *Reflection'01*, pages 170–186, 2001.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge Mathematical Textbooks, 1990.

[EAB05] Tzilla Elrad, Omar Aldawud, and Atef Bader. Expressing aspects using uml behavioral and structural diagrams. *R.E Filman, T. Elrad, S. Clarke, and M. Akżsit, editors, Aspect-Oriented Software Development, Addison- Wesley,*, pages 459–478, Boston, 2005.

[ear] Early aspects, early aspects : aspect-oriented requirements engineering and architecture design, http ://early-aspects.net, 2004.

[EFB+05] J. Estublier, J-M. Favre, J. Bézivin, L. Duchien, R. Marvie, S. Gérard, B. Baudry M. Bouzhegoub, J-M. Jézéquel, M. Blay, and M. Riveil. Action Spécifique CNRS sur l'Ingénierie Dirigée par les Modèles. Rapport de synthèse 1.1.2, CNRS, January 2005.

[Eng98] A. Engels. Message refinement : Describing multi-level protocols in MSC. In Y. Lahav, A. Wolisz, J. Fischer, and E. Holz, editors, *Proceedings of the 1st Workshop of the SDL Forum Society on SDL and MSC*, number 104 in Informatik-Berichte, pages 67–74, Berlin, Germany, June 1998. Humboldt-Universität zu Berlin.

[FF00] Robert E. Filman and Daniel P. Friedman. Aspect-oriented programming is quantification and obliviousness. In *Workshop on Advanced Separation of Concerns, OOPSLA 2000*, 2000.

[Fle06] Franck Fleurey. *Langage et méthode pour une ingénierie des modèles fiable.* PhD thesis, Université Rennes 1, 2006.

[FRGG04] Robert France, Indrakshi Ray, Geri Georg, and Sudipto Ghosh. Aspect-oriented approach to early design modelling. *IEE Proceedings Software*, pages 173–185, August 2004.

[Gen04] Blaise Genest. *LŠOdyssée des Graphes de Diagrammes de Séquences (MSC-Graphes).* PhD thesis, Université Paris 7 - Denis Diderot, 2004.

[GHM03] B. Genest, L. Hélouët, and A. Muscholl. High-level message sequence charts and projections. In *Proceedings of CONCUR'2003*, 2003.

[GSC+04] J. Greenfield, K. Short, S. Cook, S. Kent, and J. Crupi. *Software Factories : Assembling Applications with Patterns, Models, Frameworks, and Tools.* Wiley ; 1st edition, 2004. *ISBN : 0471202843.*

[HÓ0] L. Hélouët. *Analyse des exigences des systèmes répartis exprimées par des langages de scénarios.* PhD thesis, Ecole doctorale MATISSE, Universite de Rennes 1, Octobre 2000.

[Hau04] O. Haugen. Comparing uml 2.0 interactions and msc-2000. In *Proceedings of SAM 2004*, pages 69–84. LNCS 3319, 2004.

[HHC06] Loïc Hélouët, Thibaut Hénin, and Christophe Chevrier. Automating scenario merging. In *Proceedings of SAM'06*, 2006.

[HJ00] L. Hélouët and C. Jard. Conditions for synthesis of communicating automata from hmscs. In Axel Rennoch (Eds.) Stefania Gnesi, Ina Schieferdecker, editor, *5th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, Berlin, April 2000. GMD FOKUS. http ://www.gmd.de/publications/report/0091.

[HJC98] L. Hélouët, C. Jard, and B. Caillaud. An effective equivalence for sets of scenarios represented by hmscs. Technical Report 3499, INRIA, September 1998. ftp ://ftp.inria.fr/INRIA/publication/RR/RR-3499.ps.gz.

[HLM00] L. Hélouët and P. Le Maigat. Decomposition of Message Sequence Charts. In *Proceedings of SAM2000*, Grenoble, Juin 2000.

[HM03] D. Harel and R. Marelly. *Come, Let's Play : Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.

[HO93] William H. Harrison and Harold Ossher. Subject-oriented programming : A critique of pure objects. In *OOPSLA*, pages 411–428, 1993.

[HU79] JE Hopcroft and JD Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

[IEE06] IEEE. Ieee standard glossary of software engineering terminology. Technical report, IEEE, New York, September 2006.

[ITU93] ITU. *Recommendation Z.120 : Message Sequence Chart (MSC)*. E Rudolph (ed.), Geneva, 1993.

[ITU96] ITU. *Recommendation Z.120 : Message Sequence Chart (MSC)*. E Rudolph (ed.), Geneva, 1996.

[ITU99] ITU. *Recommendation Z.120 : Message Sequence Chart (MSC)*. Ø Haugen (ed.), Geneva, 1999.

[JAC] JAC. http ://jac.objectweb.org/.

[JBJC06] Andrew Jackson, Olivier Barais, Jean-Marc Jezequel, and Siobhan Clarke. Towards a generic and extensible merge operator. In *Second Workshop on Models and Aspects, Handling Crosscutting Concerns in MDSD at ECOOP 06*, Nantes, France, July 2006.

[JKBC06a] Andrew Jackson, Jacques Klein, Benoit Baudry, and Siobhan Clarke. Kertheme : Testing aspect oriented models. In *ECMDA workshop on Integration of Model Driven Development and Model Driven Testing.*, Bilbao, Spain, July 2006.

[JKBC06b] Andrew Jackson, Jacques Klein, Benoit Baudry, and Siobhan Clarke. Testing executable themes. In *Second Workshop on Models and Aspects, Handling Crosscutting Concerns in MDSD at ECOOP 06*, Nantes, France, July 2006.

[JN04] Ivar Jacobson and Pan-Wei Ng. *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley, 2004.

[KCH04] J. Klein, B. Caillaud, and L. Hélouët. Merging scenarios. In *Workshop on FMICS*, pages 209–226, Linz, Austria, sep 2004.

[KF06] J. Klein and F. Fleurey. Tissage d'aspects comportementaux. In *Langages et Modèles à Objets : LMO'06*, Nimes, France, 2006.

[KHH+01] G. Kiczales, E. Hilsdale, J. Hugunin, M Kersten, J. Palm, and W. G. Griswold. An overview of aspectj. In *ECOOP '01 : Proceedings of the 15th European Conference on Object-Oriented Programming*, pages 327–353, London, UK, 2001. Springer-Verlag.

[KHJ] J. Klein, L. Hélouët, and J.M. Jézéquel. Semantic-based weaving of scenarios. In Robert E. Filman, editor, *Proceedings of the 5th International Conference on Aspect-Oriented Software Development, AOSD 2006, Bonn, Germany, March 20-24, 2006.*

[Kic03] Gregor Kiczales. The fun has just begun. Keynote of AOSD'03, 2003.

[KL98] J.P. Katoen and L. Lambert. Pomsets for Message Sequence Charts. In *Proceedings of SAM98 :1st conference on SDL and MSC*, pages 281–290 ? ?, Berlin, Juin 1998.

[KLM+97] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[KvdBC05] Jose M. Conejero Klaas van den Berg and Ruzanna Chitchyan. Aosd ontology 1.0 - public ontology of aspect-orientation. Technical report, Technical Report AOSD-Europe-UT-01, AOSD-Europe, May, 2005.

[Lan98] Saunders M. Lane. *Categories for the Working Mathematician (Graduate Texts in Mathematics)*. Springer, September 1998.

[Lie96] Karl J. Lieberherr. *Adaptative Object-Oriented Software : The Demeter Method*. PWS Publishing, 1996.

[LW05] G.A. Lewis and L. Wrage. Approaches to constructive interoperability (cmu/sei-2004-tr-020 esc-tr-2004-020). Technical report, Pittsburgh, PA : Software Engineering Institute, Carnegie Mellon University, 2005.

[MCCV05] A. Muller, O. Caron, B. Carré, and G. Vanwormhoudt. On some properties of parameterized model application. In *In First European Conference on Model Driven Architecture - Foundations and Applications (ECMDAFA'05)*, volume 3748 of LNCS, pages 130-144. Springer, November 2005.

[MFJ05] Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving executability into object-oriented meta-languages. In *Proc. of MODELS/UML*, LNCS, Jamaica, 2005.

[MFV+05] Pierre-Alain Muller, Franck Fleurey, Didier Vojtisek, Zoé Drey, Damien Pollet, Frédéric Fondement, Philippe Studer, and Jean-Marc Jézéquel. On executable meta-languages applied to model transformations. In *Model Transformations In Practice Workshop*, Jamaica, 2005.

[MM03] J. Mukerji and J. Miller. Technical guide to model driven architecture : The mda guide v1.0.1. Technical report, OMG's Architecture Board, 2003.

[MP99] A. Muscholl and D. Peled. Message Sequence Graphs and decision problems on mazurkiewicz traces. In *Proceedings of MFCS'99*, LNCS 1672, 1999.

[MPS98] Anca Muscholl, Doron Peled, and Zhendong Su. Deciding properties for message sequence charts. In *Proceedings of FOSSACS'98*, pages 226–242. Springer-Verlag, 1998.

[MR96] Sjouke Mauw and Michel A. Reniers. Refinement in interworkings. In *International Conference on Concurrency Theory*, pages 671–686, 1996.

[Mul06] Alexis Muller. *Construction de systèmes par application de modèles paramétrés.* PhD thesis, Université de Lille 1, 2006.

[Mus99] A. Muscholl. Matching specifications for Message Sequence Charts. In *Proceedings of FOSSACS'99*, LNCS 1578, pages 273–287, 1999.

[OCL03] *UML 1.5 Object Constraint Language Specification*, March 2003. Version 1.5.

[OHCA05] Guadalupe Ortiz, Juan Hernandez, Pedro J. Clemente, and Pablo A. Amaya. How to model aspect-oriented web services. In *Workshop on Model-driven Web Engineering (ICWE'05)*, Sydney, Australia, July 2005.

[OKK+96] Harold Ossher, Matthew Kaplan, Alexander Katz, William Harrison, and Vincent Kruskal. Specifying subject-oriented composition. *Theor. Pract. Object Syst.*, 2 :179–202, 1996.

[OMG04] OMG. Meta Object Facility (MOF) 2.0 Core Specification. OMG document ptc/04-10-15, 2004. http ://www.omg.org/cgi-bin/doc ?ptc/2004-10-15.

[Omo06] Omondo, 2006. http ://www.omondo.com.

[Par72] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12) :1053–1058, 1972.

[PDF+02] Renaud Pawlak, Laurence Duchien, Gerard Florin, Fabrice Legond-Aubry, Lionel Seinturier, and Laurent Martelli. A uml notation for aspect-oriented software design. In *First Workshop on Aspect-Oriented Modeling with UML (AOSD'02)*, Enschede, The Netherlands, March 2002.

[Pra86] V. Pratt. Modeling concurrency with partial orders. *International journal of Parallel Programming*, pages 33–71, Mai, 1986.

[PSD+04] Renaud Pawlak, Lionel Seinturier, Laurence Duchien, Gerard Florin, Fabrice Legond-Aubry, and Laurent Martelli. Jac : an aspect-based distributed dynamic framework. *Softw. Pract. Exper.*, 34(12) :1119–1148, 2004.

[PSD+05] Renaud Pawlak, Lionel Seinturier, Laurence Duchien, Laurent Martelli, Fabrice Legond-Aubry, and Gerard Florin. Aspect-oriented software development with java aspect components. *R.E. Filman, T. Elrad, S. Clarke, and M. Aksit, editors, Aspect-Oriented Software Development,*, Addison-Wesley, Boston, :343–369, 2005.

[Ren98] M. Reniers. *Message Sequence Charts : Syntax and Semantics.* PhD thesis, Eindhoven University of Technology, 1998.

[RFG+05] Raghu Reddy, Robert France, Sudipto Ghosh, Franck Fleurey, and Benoit Baudry. Model composition - a signature-based approach. In *AOM Workshop*, Montego Bay, October 2005.

[RFLG04] Indrakshi Ray, Robert France, Na Li, and Geri Georg. An aspect-based approach to modeling access control concerns. *Journal of Information and Software Technology*, 46(9) :575–587, July 2004.

[RG02] Mark Richters and Martin Gogolla. OCL : Syntax, semantics, and tools. In Tony Clark and Jos Warmer, editors, *Object Modeling with the OCL : The Rationale behind the Object Constraint Language*, pages 42–68. Springer, 2002.

[RGF+06] Raghu Reddy, Sudipto Ghosh, Robert B. France, Greg Straw, James M. Bieman, Eunjee Song, and Geri Georg. Directives for composing aspect-oriented design class models. *Transactions on Aspect-Oriented Software Development (TAOSD)*, LNCS 3880 :75–105, 2006.

[RGG95] E. Rudolph, P. Graubmann, and J. Grabowski. Message Sequence Chart : composition techniques versus OO-techniques - 'tema con variazioni'. In R. Bræk and A. Sarma, editors, *SDL'95 with MSC in CASE*, Proceedings of the Seventh SDL Forum, pages 77–88, Oslo, 1995. Amsterdam, North-Holland.

[RMA03] Awais Rashid, Ana M. D. Moreira, and João Araújo. Modularisation and composition of aspectual requirements. In *proceedings of AOSD'03*, pages 11–20, 2003.

[RW94] A. Rensink and H. Wehrheim. Weak sequential composition in process algebras. In *CONCUR'94 : Concurrency Theory, 5th International Conference*, 1994.

[SHU] Dominik Stein, Stefan Hanenberg, and Rainer Unland. Expressing different conceptual models of join point selection in aspect-oriented design. In Robert E. Filman, editor, *Proceedings of the 5th International Conference on Aspect-Oriented Software Development, AOSD 2006, Bonn, Germany, March 20-24, 2006.*

[SHU02a] Dominik Stein, Stefan Hanenberg, and Rainer Unland. Designing aspect-oriented crosscutting in uml. In *First Workshop on Aspect-Oriented Modeling with UML (AOSD'02)*, Enschede, The Netherlands, March 2002.

[SHU02b] Dominik Stein, Stefan Hanenberg, and Rainer Unland. On representing join points in the uml. In *2nd International Workshop on Aspect-Oriented Modeling with UML (UML'02)*, September 2002.

[SHU02c] Dominik Stein, Stefan Hanenberg, and Rainer Unland. A uml-based aspect-oriented design notation for aspectj. In *AOSD '02 : Proceedings of the 1st international conference on Aspect-oriented software development*, pages 106–112, New York, NY, USA, 2002. ACM Press.

[SK97] Janos Sztipanovits and Gabor Karsai. Model-integrated computing. *Computer*, 30(4) :110–111, 1997.

[SKB+95] Janos Sztipanovits, Gabor Karsai, Csaba Biegl, Ted Bapty, Ákos Lédeczi, and Amit Misra. Multigraph : an architecture for model-integrated computing. In *ICECCS*, pages 361–368, 1995.

[StOs00] R. Soley and the OMG staff. MDA Model-Driven Architecture, November 2000. Online presentation http ://www.omg.org/mda/presentations.htm.

[Tar72] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2) :146–160, June 1972.

[TOHJ99] Peri L. Tarr, Harold Ossher, William H. Harrison, and Stanley M. Sutton Jr. N degrees of separation : Multi-dimensional separation of concerns. In *International Conference on Software Engineering (ICSE'99)*, pages 107–119, 1999.

[vDKV00] A. van Deursen, P. Klint, and J. Visser. Domain-specific languages : An annotated bibliography. *ACM SIGPLAN Notices*, 35(6) :26–36, June 2000.

[WA04] Jon Whittle and João Araújo. Scenario modelling with aspects. *IEE Proceedings - Software*, 151(4) :157–172, 2004.

Table des figures

Publications

Journaux Internationaux

1. ***Soumission en cours***. Jacques Klein, Franck Fleurey et Jean-Marc Jézéquel. **Weaving Multiple Aspects in Models**. *Transactions on Aspect-Oriented Software Development.*

Conférences Internationales

1. Jacques Klein, Loïc Hélouët et Jean-Marc Jézéquel. **Semantic-based Weaving of Scenarios**. *In Proc. of the 5th International Conference on Aspect-Oriented Software Development (AOSD'06)*, ACM, March 2006. Bonn, Germany (acceptance rate : 21% 20/95)

Conférences Francophones

1. Jacques Klein et Franck Fleurey. **Tissage d'Aspects Comportementaux**. *Dans les actes de Langages et Modèles à Objets (LMO'06)*, Mars 2006. Nimes, France (acceptance rate : 38% 13/34).

Workshop Internationaux

1. Jacques Klein, Benoit Caillaud, et Loic Hélouët. **Merging scenarios**. *In Formal Methods for Industrial Critical Systems (FMICS)*, vol. 133, ENTCS, pages 209–226, Linz, Austria, sep 2004 (acceptance rate : 58% 17/29).
2. Andrew Jackson, Jacques Klein, Benoit Baudry et Siobhan Clarke. **Testing Executable Themes**. *In Workshop on Models and Aspects - Handling Crosscutting Concerns in MDSD at ECOOP 2006*, Nantes, France, Jul 2006.
3. Andrew Jackson, Jacques Klein, Benoit Baudry et Siobhan Clarke. **KerTheme : Testing Aspect Oriented Models**. *In Workshop on Integration of Model Driven Development and Model Driven Testing at EC-MDA*, Bilbao, Spain, Jul 2006.
4. Jacques Klein et Jean-Marc Jézéquel. **Problems of the Semantic-based Weaving of Scenarios**. *In Aspects and Software Product Lines : An Early Aspects Workshop at SPLC-Europe 2005*, Rennes, France, sep 2005.

5. Jacques Klein, Jean-Marc Jézéquel et Noel Plouzeau. **Weaving Behavioural Models**. *In First Workshop on Models and Aspects - Handling Crosscutting Concerns in MDSD at Eccop 05*, Glasgow, United Kingdom, Jul 2005.
6. Jacques Klein et Noel Plouzeau. **Transformation of behavioral models based on compositions of sequence diagrams**. *In Proceedings of Model-Driven Architecture : Foundations and Applications 2004 (MDAFA)*, page 255, Linkoping, Sweden, jun 2004. (Poster)

Résumé

La séparation de préoccupations transverses permet au concepteur de logiciels d'avoir un meilleur contrôle sur les variations et les évolutions du logiciel. Dans le domaine de la programmation, cette idée a été popularisée par le langage AspectJ, mais aujourd'hui, la communauté aspect s'intéresse aussi à opérer cette séparation plus tôt dans le cycle de développement : dès les phases d'analyse/conception et même d'expression des besoins. Dans cette optique, cette thèse propose une technique permettant de tisser des comportements décrits sous forme de scénarios dans un modèle de base de scénarios. Le processus de tissage se décompose en deux phases. Tout d'abord, une phase de détection permettant d'identifier des parties particulières d'un modèle de base où un aspect doit être tissé, puis une phase de composition permettant de construire le modèle voulu. Ces deux phases sont détaillées dans cette thèse. En particulier, nous proposons plusieurs sémantiques de détection dans des scénarios finis. Certaines de ces sémantiques favorisent le tissage d'aspects multiples. Nous proposons également un algorithme de détection dans des scénarios infinis, fondé sur l'analyse statique de la sémantique des scénarios (qui est dynamique), ce qui revient à proposer un tissage statique d'aspects dynamiques. Pour fusionner des scénarios, nous proposons plusieurs opérateurs de composition, définis formellement, permettant d'obtenir des résultats cohérents. Finalement, nous décrivons l'implantation du tisseur de scénarios proposé dans l'environnement Kermeta, et nous montrons son utilité à travers des cas d'application.

Abstract