THESE DE DOCTORAT DE L'UNIVERSITE PARIS 1 - SORBONNE

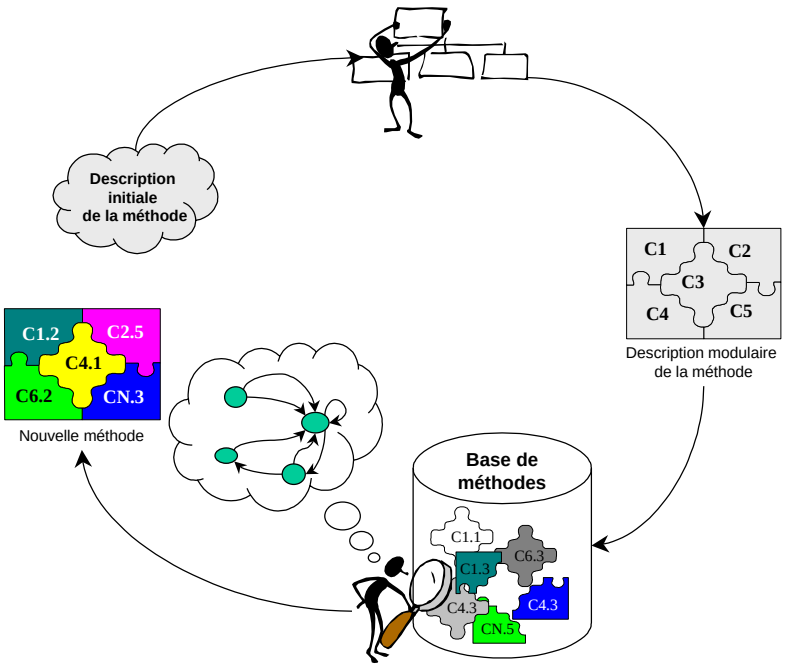
Spécialité : **INFORMATIQUE**

Jolita RALYTÉ

Pour obtenir le titre de DOCTEUR DE L'UNIVERSITE PARIS 1 – SORBONNE

Sujet de la thèse :

Ingénierie des méthodes à base de composants



Soutenu le 4 janvier 2001 devant le jury composé de :

Mme.	Colette ROLLAND	Directeur de thèse
M.	Michel LEONARD	Rapporteur
M.	Jean-Pierre GIRAUDIN	Rapporteur
Mme.	Samira SI-SAID CHERFI	
M.	Neil A.M. MAIDEN	

A ma famille

Remerciements

Tout d'abord je voudrais adresser ma profonde reconnaissance à Madame Colette Rolland, Professeur à l'Université de Paris 1 – Sorbonne qui m'a accueilli dans son équipe de recherche et a accepté d'être mon directeur de thèse. Ses idées, son savoir-faire et ses précieux conseils ont énormément contribué à ce travail de recherche.

Je suis très reconnaissante à Madame Corinne Cauvet qui a été la première à m'encourager de poursuivre mes études et m'a guidé pendant mes premières années de doctorat.

Je remercie sincèrement Monsieur Michel Léonard, Professeur à l'Université de Genève, et Monsieur Jean-Pierre Giraudin, Professeur à l'Université Pierre Mendès France, qui ont eu la gentillesse d'accepter les rôles de rapporteurs.

Je remercie Samira Si-Said Cherfi pour m'avoir encouragé durant ce travail et pour avoir bien voulu faire partie du jury de ma thèse et Neil Maiden pour avoir également accepté de faire partie de ce jury.

Je remercie tous les membres de l'équipe du Centre de Recherche en Informatique, et tout particulièrement Adolphe Benjamen, Camille Ben Achour-Salinesi, Carine Souveyet, Christophe Gnaho, Daniel Diaz, Farida Semmak, Fernando Velez, Georges Grosz, Judith Barios, Régis Kla, Selmin Nurcan qui m'ont fait l'honneur de participer par leurs efforts et leur temps, à l'élaboration de ce travail.

Je remercie tout particulièrement Rébecca Denéckère, Corinne Plourde et Mustapha Tawbi pour leur amitié, le soutien moral et l'encouragement constant durant les cinq dernières années et surtout lors de la rédaction du mémoire.

Je voudrais remercier Antans Mitašiunas, Professeur à l'Université de Vilnius, de m'avoir soutenu et encouragé dès le début de mes études en France.

Je suis très reconnaissante à Annick Cressant qui a eu le courage et la patience de relire mon mémoire. Je transmets mes amitiés à Yves et le reste de la famille Cressant.

Je remercie ma famille et tous ceux qui m'ont encouragés et aidés dans ce travail, et tout particulièrement Donatas et Laura.

Table des matières

CHAPITRE 1

Introduction

1. DOMAINE DE LA THESE

Le travail présenté dans cette thèse s'inscrit dans le domaine de l'ingénierie des méthodes et plus précisément dans le domaine de l'ingénierie des méthodes situationnelles. Celui-ci met en œuvre des méthodes qui s'adaptent aux différentes situations. Nous associons à ce domaine le principe de la réutilisation et proposons de construire des méthodes tenant compte de la situation en cours en réutilisant des fragments d'autres méthodes.

1.1 L'ingénierie des méthodes

1.1.1 Définition

De plus en plus de phénomènes de notre société sont touchés par les Systèmes d'Information (SI). La complexité des SI ne cesse de croître et par conséquent leur développement devient de plus en plus complexe, coûteux et difficile. L'utilisation de méthodes d'ingénierie pour conduire le cycle de développement d'un SI aide à mieux maîtriser la complexité des problèmes à informatiser.

Le domaine de l'ingénierie des méthodes se préoccupe de la définition de nouvelles méthodes d'ingénierie des SI. S. Brinkkemper [Brinkkemper 96] définit l'ingénierie des méthodes comme :

> *"une discipline de conceptualisation, de construction et d'adaptation de méthodes, de techniques et d'outils pour le développement des systèmes d'information"*.

Plusieurs autres définitions restreignent la notion d'ingénierie des méthodes à la construction de nouvelles méthodes à partir de celles déjà existantes. Par exemple, H.T. Punter [Punter 96] définit l'ingénierie des méthodes comme :

> *"une approche de construction des méthodes combinant différentes (parties de) méthodes pour développer une solution optimale au regard du problème donné"*.

K. Kumar [Kumar 92], au contraire, propose une définition plus générale qui n'impose pas l'utilisation des méthodes existantes comme point de départ de l'ingénierie des méthodes. Il définit cette dernière comme :

> *"une proposition pour la conception et le développement d'une méta-méthodologie destinée à la conception des méthodes de développement des systèmes d'information"*.

1.1.2 Méthode

Selon A.F. Harmsen [Harmsen 97] une méthode d'ingénierie des systèmes d'information est :

> *"une collection de procédures, de techniques, de descriptions de produit et d'outils pour le support effectif, efficace et consistant du processus d'ingénierie d'un SI"*.

Plusieurs autres définitions de la notion de méthode ont été proposées dans ([Kronlof 93], [Smolander 91], [Wynekoop 93], [Lyytinen 89], [Prakash 94], [Brinkkemper 90], [Seligmann 89], [Harmsen 94], [Brinkkemper 96]). La plupart d'entre elles convergent vers l'idée qu'une méthode est basée sur des modèles (systèmes de concepts) et consiste en plusieurs étapes qui doivent/peuvent être exécutées dans un ordre donné. Une des nombreuses définitions du concept de méthode est celle de G. Booch [Booch 91] :

> *"une méthode d'ingénierie des systèmes est un processus rigoureux permettant de générer un ensemble de modèles qui décrit divers aspects d'un logiciel en cours de construction en utilisant une certaine notation bien définie"*.

En d'autres termes une méthode est composée :

- d'un ou plusieurs modèles de produit et
- d'un ou plusieurs modèles de processus.

1.1.2.1 Modèle de produit

Le *produit* est le résultat d'application d'une méthode. Il est exprimé dans les termes d'un *modèle de produit*.

Le *modèle de produit* d'une méthode est la notation pour décrire les produits qui résultent de l'application de la méthode. Une méthode peut comporter plusieurs modèles de produit représentant chacun une facette différente des SI (structurelle, dynamique, fonctionnelle) [Olle 88] à différents

niveaux de détails (classe, paquetage, sous-système) [Muller 97], [Fowler 97], [UML 00] et à différents niveaux d'abstraction (objet, classe, méta-classe) [Shlaer 88], [Shlaer 92], [Graham 94], [Coad 91], [Rumbaugh 91], [Booch 91]. Des modèles pour la conception des entreprises sont apparus récemment et ont introduit de nouveaux concepts comme *but*, *acteur*, *rôle* [Yu 94], [Dardenne 91], [Potts 94]. Parallèlement, les ingénieurs de besoins ont introduit la distinction entre les besoins fonctionnels et non fonctionnels des SI. Par conséquent, une nouvelle typologie des modèles a été définie permettant de les classer en *fonctionnels*, *non fonctionnels* et *intentionnels* [Rolland 98c].

Un *méta-modèle de produit* est un ensemble de concepts capable de décrire tous les modèles de produit des méthodes. Il permet une représentation homogène de modèles de produit appartenant à la même méthode ou à d'autres différentes.

1.1.2.2 Modèle de processus

Le *processus* est *"un ensemble d'activités inter-reliées et menées dans le but de définir un produit"* [Franckson 91]. Il est exprimé dans les termes d'un *modèle de processus*.

Un *modèle de processus* est une démarche méthodologique décrivant la dynamique de la méthode. Le produit est le résultat d'application de cette démarche. D'une part, le modèle de processus n'a pas d'intérêt sans le modèle de produit correspondant, d'autre part la qualité du produit dépend directement de celle du processus mis en œuvre pour l'obtenir. Cependant, jusqu'à la fin des années 80, les concepteurs de méthodes se sont concentrés sur la définition des modèles de produit aux dépens de l'aspect processus. Depuis le début des années 90, on assiste à un déplacement du centre d'intérêt des modèles de produits vers la modélisation des processus d'ingénierie.

D'après Dowson [Dowson 88], les modèles utilisés pour décrire les méthodes peuvent être classés en trois types : orienté-activité, orienté-produit et orienté-décision.

Les modèles *orientés-activité* se focalisent sur les activités exécutées pour élaborer un produit et sur leur ordonnancement [Royce 70], [Boehm 88], [Curtis 88] et [Curtis 92].

Les modèles *orientés-produit* couplent l'état du produit à l'activité qui génère cet état. Ils visualisent le processus comme un diagramme de transition d'états. Le modèle des ViewPoints [Finkelstein 90] et le modèle de processus proposé dans le projet ESF (European Software Factory) [Franckson 91] appartiennent à cette catégorie.

Les modèles *orientés-décision* perçoivent les transformations successives du produit, causées par le processus comme les conséquences de prises de décisions. Les modèles de processus du projet DAIDA [Jarke 92], [Potts 89] font partie de cette catégorie. Ces modèles se focalisent sur le concept d'intention comme l'extension du concept d'activité.

Nous ajoutons deux nouvelles catégories de processus apparues récemment que nous appelons les modèles contextuels et les modèles stratégiques.

Les *modèles contextuels,* comme celui de la théorie NATURE [Rolland 95], [Plihon 96], [Jarke 99], et celui du projet F3 [Rolland 94], définissent les processus à travers la combinaison de *situations* observables avec un ensemble d'intentions spécifiques. Le travail à faire est décrit dans le processus comme étant dépendant à la fois de la situation et de l'intention. En d'autres termes, il dépend du contexte où l'on se trouve au moment de le réaliser.

La notion du *modèle stratégique* a été proposé par [Rolland 99], [Si-Said 99], [Benjament 99]. Un modèle stratégique permet de représenter les processus de développement *multi-démarche,* c'est-à-dire les processus prévoyant plusieurs chemins possibles pour élaborer le produit. Il est basé sur deux notions : l'*intention* et la *stratégie*.

Un *méta-modèle de processus* est un formalisme de description des modèles de processus. C'est un langage basé sur un ensemble de concepts qui permettent la représentation des modèles de processus.

1.1.3 Problématique

Avec la croissance de la complexité des domaines d'application, la construction de nouvelles méthodes devient de plus en plus difficile. Une méthode qui a fait ses preuves dans un domaine d'application peut être inadéquate dans d'autres domaines. La situation d'ingénierie de chaque application est différente. L'expérience a montré que les méthodes ne sont pas universelles et elles ne peuvent pas prévoir toutes les situations possibles. Les méthodes classiques ne sont pratiquement jamais suivies à la lettre. Les ingénieurs d'application sont souvent amenés à les adapter pour pouvoir les appliquer dans le contexte de leurs projets [Hidding 94]. De plus, les méthodes ne sont pas toujours suffisamment flexibles pour être facilement modifiées et adaptées.

L'ingénieur de méthodes de son coté se pose une question essentielle : comment construire une méthode qui soit assez flexible pour être facilement adaptée à la situation spécifique de chaque projet. La construction d'une méthode doit être fondée sur le principe que chaque projet est différent et que la méthode doit s'accorder à la situation.

Il existe un nombre important de méthodes disponibles pour l'ingénierie des systèmes d'information. Mais de nombreuses enquêtes [Wijers 90], [Aaen 92], [Yourdon 92], [Russo 95] ont montré que ces méthodes ne sont pas bien adaptées aux besoins de leurs utilisateurs [Lyytinen 87]. Même si les modèles de produit sont souvent bien définis dans les méthodes, les démarches qu'elles proposent sont souvent informelles et peu précisément définies [Rolland 95]. Elles sont souvent *imprécises*, *trop générales* et *mal adaptées* aux problèmes particuliers rencontrés dans la pratique, et trop *difficiles à faire évoluer*. Elles se bornent pour la plupart, à suggérer une organisation du cycle de vie en étapes globales, et ne permettent pas un guidage fin des activités de développement. Elles ne tiennent pas compte des connaissances heuristiques accumulées par les ingénieurs d'application au fur et à mesure de leur utilisation. En outre, les méthodes ne tiennent pas suffisamment compte de facteurs importants comme la situation dans laquelle le système d'information doit être développé.

Pour toutes ces raisons, les méthodes actuelles ne permettent pas aux usagers d'être guidés efficacement dans leur travail, de partager et de réutiliser leurs expériences de manière systématique.

1.2 Ingénierie des méthodes situationnelles

Les problèmes que nous venons d'évoquer, ont conduit à l'émergence d'une nouvelle approche de construction des méthodes: *l'ingénierie des méthodes situationnelles*.

L'ingénierie des méthodes situationnelles est définie dans [Welke 92a] comme :

> *"la discipline visant à construire et à adapter une méthode de développement de SI et les outils associés à chacun des projets spécifiques auxquels elle est appliquée"*.

En particulier, il est nécessaire de changer de méthode d'une situation à une autre [Hidding 94]. L'ingénierie de méthodes situationnelles [Welke 92a] est la construction de méthodes qui sont adaptées aux spécificités du projet d'ingénierie rencontré.

Une méthode doit permettre la standardisation des activités, mais elle doit cependant rester suffisamment flexible pour prendre en compte la spécificité des situations rencontrées. A.F. Harmsen [Harmsen 94] utilise le terme de flexibilité *contrôlée* pour définir cette exigence.

Cette flexibilité est obtenue dans la majorité des approches d'ingénierie des méthodes, en définissant une phase préliminaire dans le projet consistant à caractériser la situation du projet de manière globale et, à l'aide de cette caractérisation, de composer une méthode adaptée à cette situation.

De plus, le monde de la pratique des méthodes demande des processus rapides de construction de démarches définies "à la volée", pour s'adapter le mieux possible aux situations particulières de chaque projet. Le besoin de méthodes situationnelles a été introduit à ce propos [Harmsen 94], [Euromethod 94].

Ces constatations mettent en évidence le besoin de mieux comprendre la notion de méthodes, de savoir les décrire, d'être capable de les représenter, de les modéliser, mais aussi de les construire et de les utiliser dans différents contextes de projet.

L'approche de la construction des méthodes situationnelles utilise le principe de réutilisation. Les méthodes existantes sont décomposées en fragments réutilisables qui sont utilisés pour définir d'autres méthodes.

La réutilisation se définit comme une approche de développement de systèmes selon laquelle il est possible de construire un système à partir de composants existants produits à l'occasion de développements antérieurs. Cette approche s'oppose aux approches usuelles de développement dans lesquelles la construction d'un nouveau système part de zéro et nécessite de tout réinventer à chaque fois.

La réutilisation se situe à toutes les étapes du cycle de développement d'un logiciel. Introduite d'abord pour améliorer la productivité de la programmation, les tendances actuelles la transposent réutilisation à des tâches d'expression des besoins, d'analyse et de conception.

La discipline de l'ingénierie des méthodes situationnelles vise à construire des nouvelles méthodes d'ingénierie des systèmes d'information en réutilisant et assemblant différents fragments de méthodes qui ont déjà fait leurs preuves. Les quelques travaux réalisés dans ce domaine introduisent la notion de *fragment de méthode* [Seaki 94], [Rolland 96b], [Harmsen 94], [Harmsen 97] et proposent des approches d'assemblage de ces fragments de méthodes [Song 95], [Brinkkemper 98]. Selon ces approches, une méthode est vue comme une collection de fragments réutilisables. Une nouvelle méthode peut être construite en empruntant de différentes méthodes les fragments qui sont les plus appropriés pour la situation en cours [Brinkkemper 98], [Plihon 98]. Les fragments de méthodes sont alors des blocs de construction réutilisables qui permettent de définir des méthodes de manière modulaire. Les méthodes ainsi obtenues sont elles-mêmes modulaires et peuvent être modifiées et étendues facilement.

Le plus souvent, les approches de construction de méthodes par assemblage de fragments sont basées sur le regroupement de fragments de méthodes disjoints et complémentaires [Song 95], [Brinkkemper 98]. Dans notre travail nous traitons aussi l'intégration de fragments de méthodes qui se chevauchent partiellement. Notre problème peut donc être rapproché de celui de l'intégration des schémas dans le domaine des bases de données [Bouzeghoub 90], [Batini 92].

2. OBJECTIF DE LA THESE

Notre travail s'inscrit dans le domaine de l'ingénierie des méthodes situationnelles que nous venons d'introduire. L'objectif de ce travail est de proposer une approche de représentation et de construction des méthodes par assemblage de composants de méthodes ayant les propriétés suivantes:

1. *représentation modulaire des méthodes* : notre objectif est de représenter toute méthode de manière modulaire sous forme d'assemblage de modules appelés *composants de méthode*.

2. *description situationnelle et intentionnelle des composants* : chaque composant doit satisfaire un certain objectif, permettre de réaliser une activité particulière dans l'ingénierie des systèmes. Nous envisageons de définir les composants de méthodes de manière contextuelle pour pouvoir construire des méthodes situationnelles. Chaque composant doit s'appliquer dans une situation particulière pour satisfaire une intention, un objectif particulier.

3. *généricité* : l'approche doit s'appliquer à tous les types de méthodes d'ingénierie des systèmes et à tous les niveaux de cycle de développement d'un système. De plus, nous cherchons à définir les composants de méthode de telle manière qu'ils soient réutilisables indépendamment de leur méthode d'origine.

4. *réutilisabilité*: l'approche doit être basée sur le principe de la réutilisation. Tout composant de méthode doit avoir une signature qui permette de l'identifier, et un corps qui soit effectivement réutilisable.

5. *guidage du processus d'assemblage* : l'assemblage des composants de méthodes dans le but de construire une nouvelle méthode doit être guidé.

6. *organisation des composants de méthodes* : les composants définis doivent être stockés dans une base de composants de méthodes. La structure de la base doit permettre l'addition des nouveaux composants sans avoir à changer les éléments existants déjà dans la base.

3. APERÇU DE LA SOLUTION PROPOSEE

La solution proposée dans ce mémoire comporte deux parties essentielles (Figure 1):

1. la ré-ingénierie des méthodes permettant d'obtenir des méthodes modulaires et
2. la construction de nouvelles méthodes par assemblage des composants réutilisables.

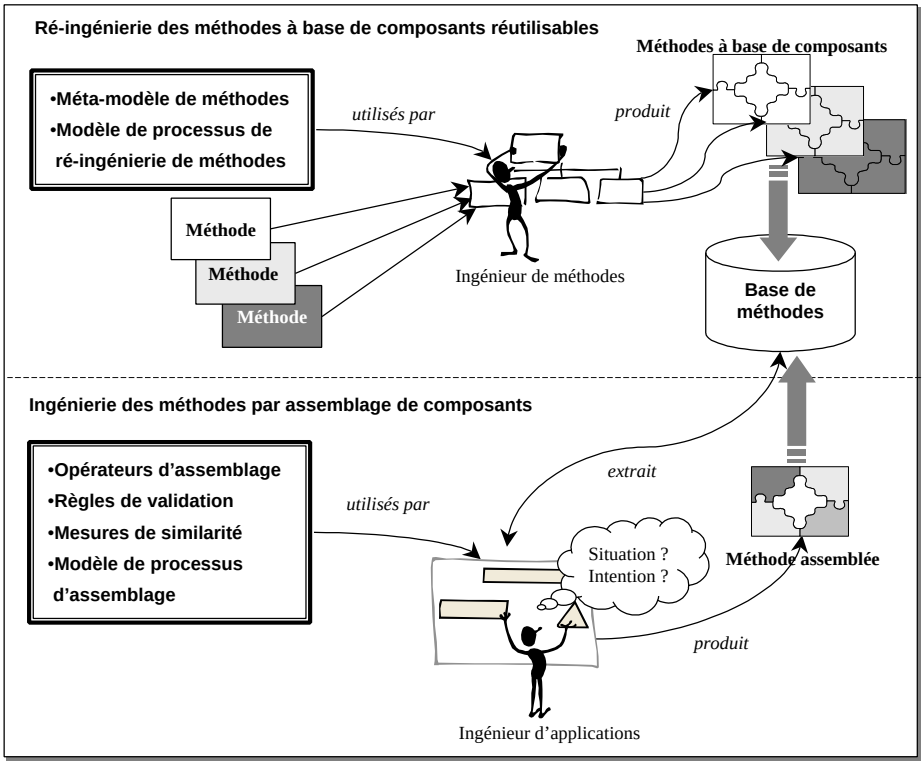


Figure 1 : Aperçue de l'approche proposée

Comme le montre la Figure 1, dans la première partie de notre solution nous proposons une approche de ré-ingénierie de méthodes existantes sous forme d'assemblages de composants réutilisables qui sont stockés ensuite dans une base de méthodes. Cette partie de notre approche comporte deux éléments :

- un *méta-modèle* permettant de modéliser des méthodes modulaires et
- un *processus de ré-ingénierie* des méthodes permettant de transformer une méthode en un ensemble de modules réutilisables appliquant le méta-modèle.

Le méta-modèle proposé dans ce mémoire permet de représenter toute méthode sous forme modulaire. Il permet de définir d'une part les modules qui composent la méthode, d'autre part, les relations entre ces modules.

Chaque module, que nous appelons *composant de méthode*, est une partie du processus proposé par la méthode, réutilisable en dehors de celle-ci. Par conséquent, la décomposition de la méthode en composants est basée sur la décomposition de sa démarche. Le modèle du processus de la méthode est décomposé en plusieurs sous-processus appelés *directives*. Une directive représentant une activité spécifique dans le développement des systèmes, réutilisable en tant qu'unité de processus indépendante et pouvant être intégrée dans différentes méthodes d'ingénierie des systèmes devient un composant de méthode. Les parties de produit sont associées aux directives dans lesquelles elles sont utilisées et participent à la définition des modèles de produit. L'ensemble, une directive et des parties de produit associées, représente un composant de méthode.

Pour obtenir le modèle de processus modulaire, nous utilisons un méta-modèle de processus appelé *carte* [Rolland 99], [Benjamen 99]. Une carte est un modèle de processus multi-démarches qui permet d'un coté, de représenter un ensemble très riche de processus puisqu'il propose plusieurs démarches en fonction des besoins des ingénieurs d'application, et de l'autre coté, il permet la représentation modulaire du modèle de processus d'une méthode.

Le méta-modèle se préoccupe également de la définition de chaque composant de méthode en tant que brique de construction de méthodes en lui associant la connaissance situationnelle et intentionnelle sous la forme d'une *signature*. Chaque composant s'applique dans une situation particulière pour réaliser une intention d'ingénierie particulière. La connaissance sur le contexte de la réutilisation de chaque composant est également définie par le méta-modèle sous forme d'un *descripteur*.

Le processus de ré-ingénierie que nous proposons est basé sur la décomposition de la carte de processus de la méthode en composants de processus réutilisables auxquels sont associés : les produit nécessaires, une signature et un descripteur permettant l'identification des composants et leur accès dans la base de méthodes.

Les composants obtenus lors du processus de ré-ingénierie sont stockés dans une base de données que nous appelons *base de méthodes*. Les composants sont facilement identifiables et accessibles dans la base grâce à la connaissance situationnelle et intentionnelle qui leur est associée.

La deuxième partie de notre approche concerne l'assemblage des composants de méthodes dans le but de construire une nouvelle méthode ou d'en enrichir une suivant les besoins actuels de l'ingénieur d'applications. Cette partie de notre approche est basée sur les quatre éléments suivants (Figure 1) :

- des *opérateurs d'assemblage* appliqués pour assembler les directives et les parties de produit des composants,
- des *règles de qualité* permettant de vérifier si les opérateurs sont appliqués correctement et de valider la cohérence et la complétude du résultat obtenu,
- des *mesures de similarité* des éléments de produit et de processus à assembler et

- un *modèle de processus d'assemblage* des composants.

L'approche que nous proposons est basée sur l'application d'opérateurs d'assemblage. Puisque chaque composant de méthode contient deux parties, la partie processus et la partie produit, nous utilisons deux types d'opérateurs : les opérateurs d'assemblage de produits et les opérateurs d'assemblage de processus. Le premier sert à assembler les modèles de produits des composants pour obtenir un modèle de produit unique. Le deuxième permet d'assembler les modèles de processus des composants pour avoir un modèle de processus unique. Une directive d'application est associée à chaque opérateur d'assemblage.

Nous proposons également dans cette approche un ensemble de règles qui sont vérifiées à chaque fois que l'on applique un opérateur ou que l'on veuille valider la complétude de la méthode obtenue.

Afin de pouvoir appliquer certains opérateurs nous devons parfois mesurer la similarité des éléments des différents composants. Le fait que deux éléments des composants différents soient similaires nous amène à appliquer certains opérateurs tandis que le contraire nous amène à en utiliser d'autres.

Finalement, nous proposons un modèle de processus d'assemblage pour guider l'ingénieur de méthodes dans la sélection et l'assemblage des composants.

Lorsqu'on applique le processus d'assemblage on obtient une nouvelle méthode qui est instance du méta-modèle proposé dans la première partie de l'approche. Par conséquent, elle est aussi une méthode modulaire et peut être stockée dans notre base de méthodes.

Notre approche s'inscrit dans les approches modulaires du spectre de méthodes de Harmsen [Harmsen 94] qui propose d'organiser les approches d'ingénierie des méthodes selon le degré de flexibilité de la au regard de la situation rencontrée. Les méthodes sont organisées sur une échelle de flexibilité variant de "faible" à "élevée".

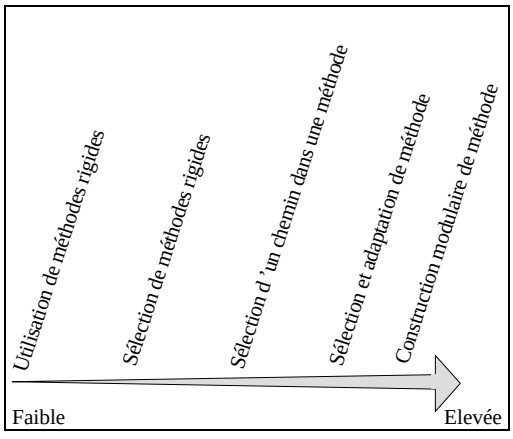


Figure 2 : Spectre des approches d'ingénierie des méthodes [Harmsen 94]

Au niveau "flexibilité faible" de ce spectre se situent les *méthodes rigides* alors qu'au niveau "flexibilité élevée", on trouve *la construction modulaire de méthode*. Les *méthodes rigides* sont complètement prédéfinies et laissent peu de possibilité d'adaptation aux nouvelles situations. A l'opposé, *les méthodes modulaires* peuvent être modifiées et améliorées pour s'adapter. La *sélection de*

méthodes rigides permet de choisir la méthode la plus adaptée à ce projet à partir d'un panel de méthodes rigides prédéfinies, tandis que la *sélection d'un chemin dans une méthode* consiste à sélectionner le chemin approprié à la situation rencontrée. Enfin *la sélection et l'adaptation d'une méthode* permettent à chaque projet de sélectionner des méthodes parmi différentes approches et de les accorder à leurs besoins.

4. PLAN DU MEMOIRE

Tout d'abord, le Chapitre 2 de ce mémoire présente l'état de l'art en ingénierie des méthodes et situe notre approche par rapport au cadre de référence que nous proposons.

Ensuite, le mémoire est organisé en deux parties principales.

La première concerne la définition des méthodes modulaires et contient deux chapitres. Le Chapitre 3 présente le méta-modèle des méthodes modulaires. Le Chapitre 4 propose le modèle de processus de ré-ingénierie des méthodes existantes.

La seconde concerne la construction de nouvelles méthodes par assemblage de composants stockés dans une base de méthodes. Elle est composée de deux chapitres. Le Chapitre 5 présente le fondement de l'approche, c'est-à-dire les opérateurs d'assemblage, les mesures de similarité et les règles de validation. Le Chapitre 6 propose le modèle de processus d'assemblage de composants.

Le Chapitre 7 de ce mémoire propose une architecture pour une base de méthodes et l'illustre avec la méthode CREWS-*L'Ecritoire*.

Le chapitre consacré à la conclusion résume l'approche proposée dans ce mémoire et propose de nouveaux développements sur ce thème.

CHAPITRE 2

Etat de l'art en ingénierie des méthodes

Ce chapitre est consacré à l'état de l'art en ingénierie de méthodes que nous présentons à l'aide d'un cadre de référence. Ce cadre est essentiellement fondé sur la notion du processus de la construction des méthodes d'ingénierie. Il est établi sous la forme d'un modèle de processus stratégique de construction de méthodes.

1. CADRE DE REFERENCE

Nous proposons un cadre de référence qui permet de classer les approches de construction de méthodes en terme de démarches de construction qu'elles proposent. Nous l'exprimons par un modèle de processus prenant en compte toutes les techniques de construction de méthodes. C'est un modèle de processus de type stratégique, que nous avons introduit au Chapitre 1, basé sur deux notions : *intention* et *stratégie*. Grâce à ces deux notions il permet d'abstraire plusieurs démarches d'ingénierie de méthodes dans un modèle de processus générique.

Le processus de l'ingénierie de méthodes peut être exprimé en terme de deux intentions principales : tout d'abord il est nécessaire de définir le propos de la construction d'une méthode, ensuite, de construire la méthode répondant à ce propos en sélectionnant une des techniques. Nous avons défini ces deux intentions de la manière suivante :

- *Définir le propos de la construction d'une méthode,*

- *Construire une méthode,*

L'intérêt d'utiliser le modèle de processus stratégique est de pouvoir proposer plusieurs stratégies différentes permettant de satisfaire les intentions. Selon la source avec laquelle on démarre la construction d'une méthode on peut classer les objectifs de la construction en deux catégories : la première concerne les cas classiques où les méthodes sont créées sans aucune source particulière ("from scrach") tandis que la deuxième prend comme source des méthodes existantes. Suivant ce raisonnement nous avons défini deux stratégies relatives à l'accomplissement de l'intention *Définir le propos de la construction d'une méthode*.

Dans le cadre de l'approche de construction adoptée pour la construction des méthodes, un certain nombre de techniques de construction est disponible. Les techniques utilisées dans le domaine des systèmes d'information ont été développées indépendamment de celles utilisées dans le domaine du génie logiciel. Dans le domaine des systèmes d'information, les techniques de construction exploitent la notion de méta-modèle et utilisent deux techniques principales qui sont l'instanciation et l'assemblage. Dans le domaine du génie logiciel la principale technique de construction utilisée est celle basée sur les langages. Cependant, les techniques utilisées dans le passé dans les deux domaines se basaient essentiellement sur l'expérience personnelle des ingénieurs de méthodes. Ces techniques étaient de ce fait de nature "ad-hoc".

La réalisation de l'intention *Construire une méthode* dépend de la technique de construction appliquée. Nous avons identifié un certain nombre de techniques existant à ce jour que nous avons exprimées en termes de stratégies :

- *ad-hoc,*
- *instanciation d'un méta-modèle,*
- *application d'un langage de modélisation,*
- *assemblage des composants,*
- *application des patrons,*
- *utilisation d'un logiciel de support.*

Une fois construites les méthodes nécessitent d'être validées. Pour cela nous proposons la stratégie d'évaluation qui prend en compte différentes techniques de validation.

Notre cadre de référence est représenté à la Figure 3.

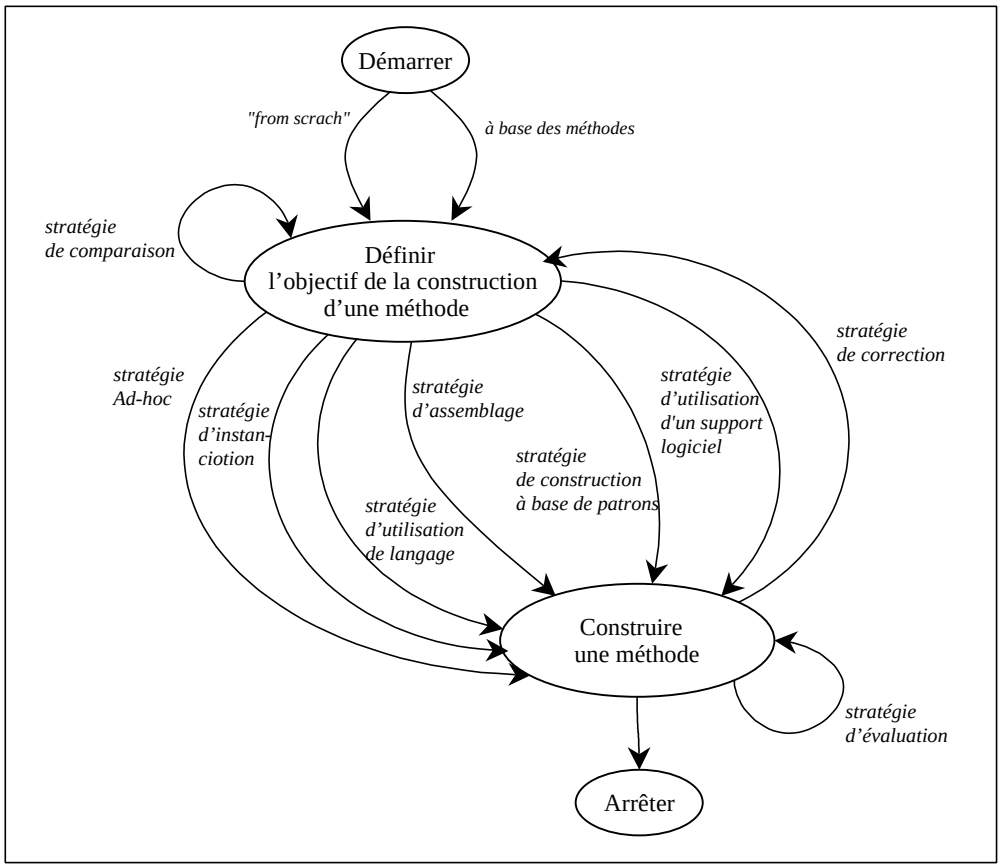


Figure 3 : Cadre de référence

La suite de ce chapitre est divisée en deux sections correspondant chacune à une des intentions de notre cadre de référence. Dans ces sections nous présentons les stratégies permettant de satisfaire ces intentions et nous classons les approches d'ingénierie de méthodes suivant les stratégies qu'elles appliquent.

2. DEFINIR L'OBJECTIF DE LA CONSTRUCTION D'UNE METHODE

Dans cette section nous analysons les objectifs que l'on cherche à atteindre dans le domaine de l'ingénierie des méthodes.

Les méthodes d'ingénierie de systèmes sont généralement supposées être indépendantes de la situation de leur application. Cependant, l'existence d'une multitude de méthodes montre que chaque méthode a des avantages et des désavantages relatifs au domaine du problème. En outre, l'expérience sur l'usage des méthodes montre que les concepteurs de système adaptent et modifient les méthodes en fonction de la situation et de leurs préférences personnelles. Les concepteurs peuvent avoir besoin de créer une nouvelle méthode à partir de rien ("from scratch"), de modifier (c'est-à-dire améliorer, compléter ou adapter) une méthode existante ou de réutiliser les parties de diverses méthodes et de les assembler pour créer la nouvelle méthode choisie.

2.1 “From scrach”

“From scratch” ou à partir de rien est une attitude classique dans la construction des méthodes. Plusieurs raisons différentes peuvent amener les ingénieurs à construire des nouvelles méthodes :

- L’expérience de conception dans un domaine particulier peut inciter le concepteur à créer sa propre méthode. Dans ce cas, la technique ad-hoc est le plus souvent utilisé. Toutefois, le concepteur peut également instancier un méta-modèle ou appliquer un langage de modélisation existant afin de décrire la méthode. De cette manière sont nées les méthodes OOSE [Jacobson 92], OMT [Rumbaugh 91] etc.

- Le fait que le concepteur ne trouve pas une méthode adaptée à son problème peut l’encourager à créer une méthode qui lui convienne. De plus, cette dernière peut être généralisée si elle a fait ses preuves lors de son application dans le projet pour lequel elle a été créée.

- Les ingénieurs peuvent avoir des nouvelles idées pour améliorer certaines activités dans le développement des systèmes. Par exemple, le projet CREWS a démontré l’utilité des scénarios dans les processus de la découverte et de la validation des besoins de systèmes [Rolland 98a] et a proposé quatre approches relatives à ces activités [Rolland 98b], [Rolland 98e], [Haumar 98], [Dubois 98], [Heymand 98], [Maiden 98a], [Sutcliffe 98a].

Tous ces objectifs peuvent être résolus en appliquant des différentes techniques de construction. L’ingénieur peut construire la méthode de manière intuitive, autrement dit ad-hoc. Cette technique, très répandue à la naissance de la discipline de l’ingénierie de méthodes, est remplacée par des techniques plus formelles et plus systématiques comme la méta-modélisation et l’instanciation du méta-modèle proposé ou l’application d’un langage formel de modélisation. Toutes ces techniques sont présentées à la section 3.

2.2 A partir de méthodes existantes

La nouvelle tendance dans l’ingénierie de méthodes est basée sur la réutilisation des méthodes existantes. L’existence d’une base de connaissance méthodologique fait émerger une large panoplie de possibilités pour construire d’autres méthodes :

- La variété et la complexité croissante des domaines à informatiser nécessitent d’adapter les méthodes existantes à chaque fois que l’on développe un nouveau système. Une adaptation peut consister à :

  – étendre une méthode par une nouvelle démarche, si celle de la méthode n’est pas assez riche,

  – améliorer la qualité de la méthode par de nouvelles propriétés (gestion du temps par exemple),

  – compléter une méthode par un nouveau modèle.

- Une idée nouvelle, encore plus puissante, est née dans les années 90 en proposant de construire des méthodes en réutilisant les différentes parties de plusieurs méthodes. Ceci permet la construction d'une méthode "à la volée".

- Un autre objectif de l'ingénierie des méthodes est d'assembler des composants de méthodes et de construire des outils CAME pour la ré-ingénierie des méthodes existantes. Elle peut consister à :

  – décrire la méthode d'une manière formelle en appliquant un langage particulier ou

  – reconstruire la méthode sous forme de composants réutilisables.

- L'expérience en appliquant une méthode peut faire apparaître des erreurs qui nécessitent d'être corrigées.

Tous ces objectifs peuvent être satisfaits en appliquant des nouvelles techniques de construction comme l'assemblage, l'application des patrons génériques ou l'utilisation des outils CAME. La plupart de ces techniques sont basées sur les techniques comme la méta-modélisation et l'application des langages formels. Ces techniques sont présentées à la section 3.

2.3 Comparaison des méthodes

Pour évaluer les méthodes et pour aider dans la sélection des méthodes, des techniques de comparaison ont été proposées. Par exemple, Iivary propose un cadre de référence pour comparer les méthodes orientées-objet qu'il applique à six méthodes [Iivary 94]. Ce cadre divise le processus de développement des systèmes d'information en trois niveaux : organisationnel, conceptuel et technique. Chaque niveau est analysé sous trois vues différentes : structurelle, fonctionnelle et comportementale. Dans le projet TOOBIS Souveyet et al. s'inspire du travail de Iivary pour comparer des méthodes oriéntées-objet [Souveyet 97]. Dans ce travail le cadre de référence de Iivary est complété par des relations entre les trois niveaux et les trois vues.

[Song 92] et [Hong 93] proposent des approches pour comparer des méthodes d'une manière systématique. Ces approches sont basées sur la méta-modélisation des méthodes à comparer. Elles proposent de construire tout d'abord les méta-modèles des toutes les méthodes à comparer en utilisant le même formalisme, ensuite elles se servent de ces méta-modèles pour comparer différents aspects de ces méthodes. [Hong 93] par exemple, compare les méthodes selon trois perspectives : les étapes d'analyse et de conception, les concepts et les techniques. Une représentation tabulaire est utilisée par les deux approches pour comparer les méthodes.

[Rossi 95] propose une approche qui permet de mesurer la complexité des méthodes d'une manière systématique et automatisée. Cette approche utilise un ensemble de métriques permettant de mesurer d'un coté la difficulté de comprendre et d'apprendre la méthode causée par le nombre de différents concepts utilisés dans la méthode et d'un autre coté la complexité des produits causée par le nombre de propriétés des objets et des relations. Ces métriques peuvent être utilisées au moins pour deux

objectifs : premièrement par l'ingénieur de méthodes pour valider les propriétés de la méthode, deuxièmement par l'utilisateur de méthodes pour sélectionner des méthodes.

3. CONSTRUIRE UNE METHODE

Suivant notre cadre de référence présenté à la Figure 3, nous présentons dans cette section six techniques de construction de méthodes : *ad-hoc*, *instanciation d'un méta-modèle*, *application d'un langage de modélisation*, *assemblage*, *application des patrons génériques* et *utilisation d'un outil CAME*, puis un résumé des approches d'évaluation des nouvelles méthodes.

L'approche que nous proposons dans cette thèse s'inscrit dans la catégorie des techniques d'assemblage. La section décrivant cette technique comporte la comparaison de notre approche avec d'autres approches appartenant à la même catégorie.

3.1 Ad-Hoc

La technique de construction de méthodes *ad-hoc* est une technique traditionnelle qui est basée sur l'expérience acquise lors des développements de différents systèmes d'un domaine spécifique. Tant que cette expérience n'est pas formalisée et ne constitue pas une connaissance de base disponible pour les différents ingénieurs d'application, on peut dire que cette connaissance est le résultat d'une technique de construction ad-hoc. Ceci a deux conséquences majeures : la méconnaissance de la manière dont la méthode à été générée et sa dépendance au domaine d'expertise. Si la méthode doit être indépendante du domaine d'expertise, facile à appliquer et à modifier, il est alors nécessaire de sortir du cadre des techniques de construction basées sur l'expérience. Les techniques comme l'instanciation et l'assemblage favorisent la flexibilité et la modularisation des méthodes et facilitent la capitalisation des bonnes pratiques et l'amélioration des modèles existants.

3.2 Instanciation d'un méta-modèle

La technique de construction des méthodes par intanciation est basée sur la méta-modélisation.

La méta-modélisation consiste à identifier les caractéristiques communes et génériques des différents modèles et à les représenter ensuite par un système de concepts génériques. Une telle représentation, appelée méta-modèle, permet de générer tous les modèles partageant ces mêmes propriétés. Cette technique de génération doit être définie de telle manière qu'elle produise le modèle désiré. Pour résumer, les deux problèmes suivants doivent être résolus :

- l'identification d'un système de concepts génériques inter reliés,
- la définition des techniques d'instanciation.

Le premier problème est résolu par la définition d'un *méta-modèle de méthodes* alors que le second est résolu par la dérivation des modèles de ce méta-modèle à travers son *instanciation.*

3.2.1 Méta-modélisation

A ce jour on peut dire que les recherches les plus intensives dans le domaine de l'ingénierie de méthodes ont été menées dans le domaine de la méta-modélisation. On peut même dire que la méta-modélisation est le fondement de l'ingénierie de méthodes.

Plusieurs méta-modèles de méthodes ont été proposés. La plupart d'entre eux prennent en compte les deux perspectives de modélisation : le produit et le processus [Saeki 93], [Plihon 96], [Prakash 99]. La perspective produit définit les structures des produits qui sont construits lors de l'application de la méthode tandis que les activités réalisées pour construire ces produits sont modélisées dans la perspective processus de la méthode.

Un certain nombre de méta-modèles de produit sont basés sur le modèle sémantique proposé par Hull et King [Hull 87]. Mais les plus utilisés sont les modèles basés sur les formalismes de ER et de NIAM. Toutefois, il est difficile de représenter les contraintes et les structures hiérarchiques des méthodes en utilisant ces formalismes. Des extensions de ER proposées dans [Sorenson 88], [Welke 92b], [Smolander 91] cherchent à améliorer le pouvoir d'expression en prenant en compte les contraintes d'intégrité et la représentation des objets complexes. Les méta-modèles de produit basés sur les formalismes de NIAM [Bommel 91], [Hofstede 93], [Hofstede 93a] aboutissent à des résultats similaires mais sont fondés sur une base plus formelle que les précédents.

Un autre formalisme de méta-modélisation a été adapté du langage de modélisation orienté-objet appelé Object-Z par Seaki et Wenyin [Seaki 94a]. Ahituv [Ahituv 87] propose un méta-modèle formel qui voit un système d'information comme un flux de données qui progresse d'un état vers un autre.

L'aspect processus dans la méta-modélisation est souvent développé séparément de celui de produit. Dowson [Dowson 87] distingue trois catégories dans la modélisation des processus : orienté-activité, orienté-produit et orienté-decision. Nous les avons introduits au Chapitre 1 de ce mémoire et y avons ajouté deux nouvelles catégories récentes : les modèles de processus contextuels et les modèles de processus stratégiques.

Les méta-modèles de processus comme *Cascade* [Royce 70], *Spirale* [Boehm 90], *Hiérarchique en spirale* [Iivari 90] ou *Fontaine* [Henderson-Sellers 90] font partie de la catégorie des modèles orienté-activité. Ces méta-modèles permettent de modéliser le processus par un ensemble de phases de haut niveau organisées en un processus séquentiel. Ils s'adressent donc à des processus tactiques dont l'objectif est de planifier les étapes à suivre.

Les modèles de processus orienté-produit représentent le processus de dévelopement à travers l'évolution du produit qui en est le résultat. Ils ne mettent plus en avant les activités du processus, mais le résultat de ces activités et le produit résultant. Les méta-modèles de processus VIEWPOINTS [Finkelstein 90] et ESF [Franckson 91] appartiennent à cette catégorie. Dans la même catégorie le méta-modèle de processus de [Humprey 89] permet de visualiser les processus par des diagrammes de

transition d'états du produit en construction. [Tomiyama 89] et [Akman 90] voient les processus comme l'évolution du méta-modèle du produit de conception.

Les modèles de processus orienté-decision expriment les transformations de produit comme conséquences de décisions. Les méta-modèles de processus proposés dans les projets IBIS [Pots 89] et DAIDA [Jarke 92] appartiennent à cette catégorie. De tels modèles sont sémantiquement plus riches que les précédents car ils permettent d'expliquer non seulement comment le processus se déroule mais aussi pourquoi les actions sont exécutées.

On classe le méta-modèle de processus proposé par le projet NATURE [Rolland 95], [Plihon 96], [Grosz 97], [Jarke 99] dans la catégorie des processus contextuels. Selon ce méta-modèle, à tout moment le comportement de l'ingénieur d'application est conditionné par un contexte particulier : il se situe dans une situation particulière définie par l'état actuel du produit en construction où il a une intention à réaliser. Cette intention fait partie de l'objectif général - construire le produit.

Récemment une toute nouvelle catégorie de processus est apparue avec la proposition de Rolland et al. dans [Rolland 99] et Benjamen dans [Benjamen 99]. Il s'agit d'un nouveau méta-modèle de processus appelé *Carte*. Nous avons appelé cette catégorie *processus stratégique* à cause de son principe de modélisation des processus en termes d'*intentions* d'ingénierie à réaliser afin de construire le produit et de *stratégies* à suivre pour réaliser ces intentions. Ce méta-modèle a déjà fait ses preuves dans le projet CREWS[1] où il a été utilisé pour modéliser quatre approches d'ingénierie de besoins à base de scénarios. Nous l'exploitons également dans les travaux présentés dans ce mémoire.

La naissance d'un nouveau domaine dans l'ingénierie de méthodes, l'ingénierie des méthodes situationnelles, a stimulé la parution de méta-modèles permettant de représenter des méthodes sous forme modulaire. Rolland et Prakash ont proposé une structure pour un composant de méthodes dans [Rolland 96]. Dans [Prakash 97] et [Prakash 99] une méthode est vue comme un ensemble de blocs de méthode, chaque bloc incorporant l'aspect produit et l'aspect processus. Le méta-modèle de méthodes que nous avons introduit dans [Rolland 98a] et [Ralyte 99b] et que nous détaillons dans ce mémoire permet de représenter toute méthode sous forme d'un assemblage de composants de méthodes. Chaque composant obtenu en appliquant notre méta-modèle devient un module réutilisable dans la construction des méthodes par assemblage de composants.

3.2.2 Instanciation

La technique d'instanciation d'un méta-modèle de processus contextuel a été utilisée dans [Rolland 93], [Rolland 94a], [Rolland 94b], [Rolland 96a]. L'ingénieur doit définir les instances de contextes et de relations entre les contextes qui composent le modèle en question. Cette technique a également été utilisée pour construire les bases de connaissance d'Atelier d'Ingénierie des Méthodes [Kelly 96], [Harmsen 95], [Merbeth 91], [Si-Said 96].

[1] CREWS – Co-operative requirements Engineering With Scenarios

La technique de construction des méthodes par instanciation peut être résumée par la Figure 4.

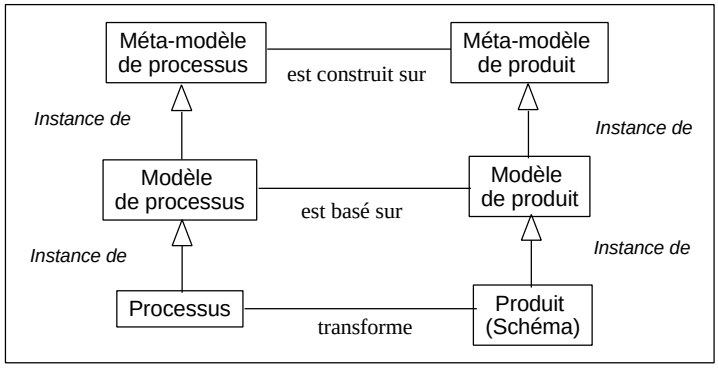


Figure 4 : Les niveaux d'abstraction

Au niveau le plus haut, se trouvent le *méta-modèle de processus* et le *méta-modèle de produit*. Ces deux méta-modèles contiennent des concepts génériques qui permettent respectivement, de décrire le *modèle de produit* et le *modèle de processus* d'une méthode spécifique.

Tout élément d'un modèle de produit est instance d'un concept du *méta-modèle de produit*. Le *méta-modèle de produit* définit les concepts nécessaires pour caractériser une situation du produit. De même, tout élément du *modèle de processus* est instance d'un concept du *méta-modèle de processus*. En généralisant, nous dirons qu'un *modèle de produit* est construit par instanciation du *méta-modèle de produit* et qu'un *modèle de processus* est construit par instanciation du *méta-modèle de processus*. Les concepts du modèle de processus font référence aux concepts du modèle de produit.

Au niveau d'une application réelle, l'ingénieur d'application suit un *processus.* L'exécution de ce processus est contrôlée par une instance du *modèle de processus* qui prescrit la marche à suivre. Le *processus* exécuté transforme un *produit* dont les éléments sont des *instances* du *modèle de produit.*

3.2.3 Conclusion

Les avantages d'utilisation de la technique méta-modélisation-instanciation sont les suivants :

1. L'exploitation du méta-modèle aide à définir un certain nombre de modèles.

2. Cela permet de définir des composant de méthodes de manière systématique.

3. Cela oblige à chercher et à introduire dans un méta-modèle des solutions génériques aux problèmes traités. Les modèles ainsi dérivés hériteront des caractéristiques de la solution. A travers l'approche par instanciation, le problème crucial n'est plus celui des modèles mais celui du méta-modèle de processus. Cela signifie que la responsabilité de fournir un bon modèle de processus n'incombe plus au concepteur de modèles mais au concepteur du méta-modèle.

La méta-modélisation est non seulement utile dans la construction des méthodes mais aussi dans la formalisation des méthodes mal définies [Tolvanen 93], dans la comparaison des méthodes [Hong

93], [Rossi 96], dans la standardisation des méthodes [Booch 97], [OMG 97] et dans la définition des liens entre les méthodes d'ingénierie et les langages de programmation [Hillegersberg 97].

3.3 Application d'un langage de modélisation

Brinkkemper [Brinkkemper 90] fait l'hypothèse que tout langage conceptuel de modélisation puisse servir de langage de méta-modélisation. Divers langages de méta-modélisation d'application destinés à l'origine à d'autres domaines, tels que LOTOS [Saeki 91] semblent démontrer la validité de cette hypothèse. Cependant d'autres auteurs ont des revendications contraires. Venable et Grundy expriment le fait qu'il y a besoin d'un langage de méta-modélisation et d'un environnement spécifiques. Le langage COCOA [Venable 93] a été développé et employé dans l'environnement MVIEWS [Grundy 96].

Les exigences générales sur l'ingénierie de méthodes définies dans [Kottemann 84], [Marttiin 95], [Welke 92a] soulignent la nécessité de constructions sémantiques assez riches pour modéliser la structure conceptuelle et les contraintes des méthodes.

La discipline de l'ingénierie des méthodes devenant de plus en plus mature, plusieurs écoles proposant des différents langages de représentation et de manipulation de méthodes ont vu le jour. Selon Saeki [Saeki 96], on peut distinguer quatre écoles principales. La première opte pour une approche orientée-données accentuant la représentation de l'aspect produit des méthodes. Elle inclut les langages d'ingénierie de méthodes comme GOPRR [Smolander 92], [Kelly 96], PSM-LISA/D [Hofstede 93], les diagrammes de structure objet de NIAM [Brikkemper 90], [Wijers 91], les modèles sémantiques de données [Sowa 92] et ASDM [Heym 92]. Une deuxième école adopte l'approche orientée-objet. Les langages appartenant à cette école sont Telos [Mylopoulos 90], Mataview [Sorenson 88] et ObjectZ [Saeki 94a]. La troisième école concerne les langages développés pour modéliser des processus de construction de logiciels et saisir les motivations de la conception. Les langages comme les Diagrammes de Structure de Tâches [Wijers 91], [Verhoef 95], HFSP [Katayama 89], [Song 92], ALF [Benali 89] et MERLIN [Emmerich 91] appartiennent à ce groupe. La dernière école concerne les langages, appelés "langages hybrides", qui prennent en compte les différent aspects de l'ingénierie de méthodes. Le langage MEL proposé par Harmsen et al. [Harmsen 95a], [Harmsen 95b] et Brinkkemper dans [Brinkkemper 95] fait partie de cette catégorie. MEL est un langage de la spécification et de la manipulation pour la construction des méthodes situationnelles.

Harmsen et Saeki [Harmsen 96] ont choisi quatre langages (un de chaque catégorie) : Object-Z, MEL, GOPRR et HFSP, et les ont comparés. La conclusion de l'étude est qu'il n'y a pas de langage universel pour l'ingénierie de méthodes. Les différents langages ont des propriétés communes et des propriétés complémentaires. Le choix du langage dépend de l'objectif à atteindre dans l'IM. Comme solution, Harmsen et Saeki proposent d'appliquer la technique d'assemblage sur les langages d'ingénierie de méthodes eux-mêmes et de construire ainsi un langage adapté à la situation donnée à partir de divers fragments de langages existants.

Les langages de méta-modélisation nécessitent différentes formes de représentations. Ils utilisent traditionnellement des notations informelles comme le langage naturel ou des diagrammes avec des

sémantiques informelles. Le fait que les méthodes utilisent souvent des représentations schématiques a influencé le développement visuel des langages de méta-modélisation. Weide dans [Weide 93b] accentue l'importance de la représentation graphique lors de l'instanciation d'un méta-modèle afin d'obtenir une méthode et propose une approche de la formalisation d'une telle représentation. Des langages pour représenter les notations graphiques des méthodes, les connexions entre celles-ci et les contraintes graphiques sont proposés dans [Sommerville 87], [Smolander 91], [Protsko 89], [Hofstede 92]. Kelly [Kelly 94] et Kinnuenen et al. [Kinnuenen 94] ont investi dans l'utilisation des matrices comme moyen de représentation de la méta-modélisation. Les dernières recherches dans ce domaine étendent les représentations précédentes dans de nouveaux paradigmes multi-représentations. Prenons comme exemple, l'environnement MetaEdit+ qui permet de modéliser une méthode en utilisant un diagramme, une matrice ou un tableau et qui propose aussi un mode graphique pour questionner la base de méthodes [Liu 95].

Les travaux présentés dans [Oei 92], [Oei 94], [Oei 95] introduisent un langage formel pour modéliser les méthodes et les organiser dans une hiérarchie.

3.4 Assemblage

La technique de construction de méthodes par assemblage est la dernière-née du domaine d'ingénierie de méthodes. Plus précisément, elle est apparue avec l'arrivée de l'ingénierie de méthodes situationnelles au début des années 90. Cette dernière a apporté une nouvelle tendance dans l'ingénierie de méthodes – la construction des méthodes spécifiques à des projets [Welke 91].

Selon le spectre de méthodes proposé par Harmsen dans[Harmsen 94] qui organise les approches de construction de méthodes situationnelles selon leur degré de flexibilité face aux situations des projets rencontrés (voir le Chapitre 1), la technique d'assemblage a le degré de flexibilité le plus élevé.

Cette technique est fondée sur l'idée que l'on peut décomposer les méthodes existantes en composants (aussi appelés fragments, modules ou blocs) réutilisables indépendamment. La construction d'une nouvelle méthode consiste donc à sélectionner des fragments de différentes méthodes correspondant à la situation d'un projet en cours et à les assembler. Ceci fait apparaître les problèmes suivants:

- la redéfinition des méthodes existantes sous forme de composants,
- le stockage de ces composants dans une base de méthodes et
- le guidage dans la sélection et l'assemblage des composants.

Le travail que nous proposons dans cette thèse utilise la stratégie de construction de méthodes par assemblage des composants. Afin de pouvoir comparer notre approche avec les approches existantes appartenant à la même catégorie nous présentons en quelques mots certaines parmi elles.

- Harmsen, Brinkkemper et Saeki proposent dans [Harmsen 94], [Harmsen 95], [Brinkkemper 98] une approche d'assemblage à base des fragments de méthodes de différents niveaux de

granularité. Afin de le distinguer parmi les autres approches nous lui avons donné le nom de *Fragments de méthodes.*

- Rolland et Praksh proposent une structure pour représenter et stocker des composants de méthodes de différents niveaux d'abstraction et de différents niveaux de granularité. Nous avons appelé cette approche *Composants contextuels.* Elle est présentée dans [Rolland 96].

- Prakash propose un méta-modèle pour représenter des méthodes sous forme modulaire [Prakash 97], [Prakash 99]. Les modules sont appelés des blocs de méthodes d'où le nom de l'approche *Blocs de méthodes.*

- Song propose une approche d'intégration des méthodes de conception qui est basée sur l'adaptation des méthodes existantes [Song 97]. Elle permet de compléter une méthode par une fonctionnalité complémentaire prise dans une autre méthode et/ou d'améliorer sa qualité en lui associant des propriétés issues d'autres méthodes.

- Punter et Lemmen proposent une approche appelée MEMA [Punter 96] comportant un méta-modèle pour représenter les fragments de méthodes ainsi qu'un processus de sélection et d'assemblage de fragments afin de construire une méthode pour un projet concret.

- L'approche proposée par Van Slooten se base surtout sur la caractérisation des projets en utilisant des facteurs de contingence. [Van Slooten 93], [Van Slooten 96].

Nous analysons maintenant comment ces approches réagissent aux problèmes dont la liste a été donnée plus haut dans ce paragraphe.

3.4.1 Structure de composant

Dans la plupart des cas la définition des composants est basée sur la technique de méta-modélisation. Ceci permet de résoudre le problème de représentation des méthodes sous forme de composants. La plupart des approches existantes ne prennent en compte que la structure des composants et ne se préoccupent pas du processus de reconstruction des méthodes sous forme de composants.

Pratiquement chaque approche propose plusieurs types de composants qu'elle classifie selon différentes typologies. Par exemple, Brinkkemper et al. [Brinkkemper 98] classifie les composants selon trois dimensions : la *perspective*, l'*abstraction* et la *granularité*, que nous utilisons comme fondement de classification.

3.4.1.1 Dimension de perspective

La dimension de *perspective* considère les méthodes du point de vue du *produit* et de celui du *processus.*

Certaines approches font une séparation entre la perspective produit et la perspective processus et proposent deux types de composants : ceux du produit et ceux du processus. Dans cette catégorie nous

retrouvons l'approche *Fragments de méthodes* qui définit des fragments de produit et des fragments de processus. Les fragments de produit représentent des modèles, des diagrammes, des documentations, etc. Les fragments de processus représentent des étapes, des activités des tâches à réaliser. Chaque fragment de processus a un fragment de produit associé et inversement.

L'approche proposée dans [Song 97] fait également la séparation entre la perspective produit et la perspective processus. Dans la première perspective elle propose deux types de composants appelés *modèle artefact* et *composant de modèle* appartenant à des niveaux de granularité différents. Dans la deuxième elle propose deux types de composants appelés *processus* et *action* qui appartiennent également à deux niveaux de granularité différents.

D'autres approches s'appuient sur le fait que la perspective de processus ne peut pas être séparée de celle de produit, que le processus est toujours basé sur le produit qu'il manipule. Par conséquent, elles affirment qu'il est préférable de coupler les deux perspectives dans le même composant de méthode. Dans cette catégorie on trouve les composants de Rolland et Prakash décrits dans [Rolland 96], l'approche *Blocs de méthodes* de Prakash, l'approche MEMA de Punter ainsi que les composants de méthode décrits dans [Rolland 98], [Ralyte 99b]et le Chapitre 3 de ce mémoire.

3.4.1.2 Dimension d'abstraction

Selon Brinkkemper [Brinkkemper 98], la dimension d'*abstraction* est constituée de deux niveaux : *conceptuel* et *technique*. Le niveau conceptuel comporte des fragments de méthodes de conception tandis que le niveau technique propose des fragments qui représentent des spécifications implémentables des parties opérationnelles de méthodes, c'est-à-dire des outils.

La plupart des approches ne sont considérées que dans un des niveaux d'abstraction – celui de conception. Parmi ces approches figurent *Blocs de méthodes*, MEMA, *Composants contextuels*.

L'approche *"Fragments de méthodes"* prend en compte les deux niveaux d'abstraction en précisant que chaque fragment technique correspond à un fragment conceptuel.

Dans notre approche les parties techniques, si elles existent, ne sont pas dissociées des parties conceptuelles correspondantes. Autrement dit, si un composant de méthode est assisté par un outil, ce dernier est attaché au composant de méthode.

[Roland 96a], [Roland 96b] et [Rolland 97] a un autre point de vue sur les différents niveaux d'abstraction de composants. Selon son niveau d'abstraction, le composant de méthode est réutilisé en tant que tel ou doit être instancié pour être assemblé dans la méthode en construction. L'approche [Rolland 96b] propose trois types de composants : *cadre de référence*, *patron de construction* et *composant de méthode*. Le *cadre de référence* est un composant qui formalise d'une manière abstraite la connaissance commune à plusieurs méthodes. Le patron modélise le comportement commun dans la construction des méthodes. Il est générique dans le sens qu'il est utilisé par un ingénieur de méthodes dans le processus de construction de chaque nouvelle méthode. Par conséquent, le composant *patron* est plus abstrait que le *cadre de référence*. Ces deux termes ont été choisis par analogie avec les approches de réutilisation dans le domaine orineté-objet. Les patrons dans ce domaine sont définis

comme des solutions aux problèmes génériques qui apparaissent dans plusieurs applications [Gamma 93], [Pree 95] (voir la section 3.6) tandis que les cadres de référence sont dépendants du domaine d'application [Wirfs-Brock 90], [Johnson 88].

3.4.1.3 Dimension de granularité

La dimension de *granularité* est la plus importante et la plus discriminante des classifications proposées. Elle est basée sur la décomposition des méthodes en différents niveaux de détail. Par exemple, du point de vue du processus une méthode peut être organisée en étapes qui, à leur tour, sont structurés en activités qui sont décomposées en actions etc. Une décomposition similaire peut également être appliquée sur le produit de la méthode car celle-ci peut être représentée par un ensemble de modèles ayant différents diagrammes, qui à leur tour sont décomposés en concepts, etc.

L'approche *Fragments de méthodes*, propose cinq niveaux de granularité de fragments appelés: *méthode*, *étape*, *modèle*, *diagramme* et *concept*.

- Le niveau ***méthode*** s'adresse à des méthodes entières. Toute méthode d'ingénierie de systèmes, comme par exemple OMT, OOSE, est vue comme un fragment de méthode.

- Le niveau ***étape*** s'adresse à des segments dans le cycle de vie d'un système d'information. Par exemple, *Analyse de domaine*, *Outil CASE*, *Rapport technique de la conception d'un système*, sont des fragments de niveau étape.

- Un fragment de type ***modèle*** prend en compte une perspective d'un système d'information comme *Modèle de donnés*, *Modèle d'interface utilisateur* etc.

- Un fragment de type ***diagramme*** correspond à une représentation possible d'un composant de type modèle. Par exemple, un *Diagramme d'objets* et un *Diagramme de classes* sont deux représentations possibles du fragment *Modèle de données*.

- Le niveau ***concept*** s'adresse à des concepts et à des associations qui existent entre eux dans le niveau *diagramme* d'une méthode ainsi que les manipulations qui peuvent être faites avec eux. *Entité*, *Relation entre deux entités* et *identifier une entité* sont des exemples des fragments de niveau concept.

L'approche [Song 95] divise tous les composants de méthode en deux niveaux de granularité qu'elle appelle : le *niveau haut* et le *niveau bas*. Le niveau haut correspond au niveau des modèles de méthodes tandis que le niveau bas représente les différents éléments de ces modèles. De plus, cette approche sépare non seulement les notions de produit et de processus, mais en outre leurs représentations et leurs propriétés sont considérées à part dans des composants spécifiques.

Le niveau *haut* propose cinq types de composants : les ***modèles artefacts*** qui représentent des structures, des modèles (par exemple le modèle objet de la méthode OMT), les ***propriétés*** qui considèrent les caractéristiques des modèles artefacts, les ***principes*** qui décrivent des règles devant être suivies en construisant des artefacts (par exemple les principes d'abstraction et d'encapsulation

dans les méthodes orientées-objet), les ***représentations*** concernent le moyen d'expression des modèles artefacts (par exemple le diagramme d'objet ou le diagramme de flux de donnés) et les ***processus*** qui décrivent des étapes que le concepteur doit utiliser dans le développement d'un système en appliquant un modèle artefact.

Les composants appartenant au niveau *bas* sont partagés en six catégories : les ***modèles composants*** qui sont des éléments d'un *modèle arteface* (par exemple, une classe dans le modèle objet de la méthode OMT), les ***critères*** qui sont des règles que le concepteur doit appliquer pour décider si un artefact est une instance d'un modèle composant (par exemple, l'ingénieur d'application devrait utiliser ces règles pour décider si l'artefact « porte » peut être une classe dans le système de contrôle d'ascenseur), les ***directives*** qui sont des stratégies, des heuristiques ou des techniques concrètes pour identifier et décrire les artefacts, les ***mesures*** qui concernent les aspects des artefacts qui peuvent être mesurés, les ***notations*** qui sont des parties de la représentation utilisée pour exprimer les artefacts (par exemple une boite pour représenter les objets dans le diagramme objet de OMT), les ***actions*** qui sont des étapes pour développer les artefacts (une action peut créer, modifier, utiliser ou évaluer un artefact).

Les différents composants appartenant au même niveau sont reliés entre eux par des liens d'association tandis que les composants de niveaux différents sont reliés par des liens de composition.

Van Slooten dans [Van Slooten 93], [Van Slooten 96] propose deux types de composants de méthodes appartenant à deux niveaux de granularité, appelés la *carte de routes* et le *fragment de méthode*. Une *carte* de *routes* est un plan représentant une stratégie de développement. Il est composé d'activités de développement et de produits concernant le développement d'un système ainsi que la gestion du projet. Par exemple, une carte de routes peut représenter une stratégie de planification du projet, une stratégie de fourniture du projet à l'utilisateur (en entier, incrémentale, évolutive), une stratégie de réalisation, une stratégie d'organisation du projet, une stratégie de la gestion du projet etc. Un *fragment de méthode* est une partie cohérente d'une méthode, d'une technique ou d'un outil destiné au développement d'un système ou à la gérance d'un projet. Les fragments de méthodes peuvent être incorporés dans la carte de routes pour établir une approche complète pour un projet.

Dans notre approche nous avons également des composants à différents niveaux de granularité. Nous distinguons deux types de composants : atomiques et agrégats. Les composants atomiques peuvent être reliés par des liens de composition, d'alternative ou d'imbrication afin de construire des composants plus importants que l'on appelle des agrégats.

3.4.2 Stockage de composants

Le deuxième problème concernant la technique d'assemblage est le stockage de composants et l'accès aux composants.

La plupart des approches d'assemblage utilisent la notion de base de méthodes pour stocker les composants. On trouve une base de méthodes dans les approches *Fragments de méthode*, *Composants*

contextuels, MEMA, celle de van Slooten ainsi que la nôtre. Toutefois la plupart des approches ne précisent pas quelle est la structure de cette base ni comment les composants peuvent y être atteints.

[Song 97], au contraire, ne parle pas d'une base de méthodes et ne dit pas comment décrire ni comment stocker les composants. L'approche *Blocs de méthodes* propose un formalisme pour décrire les blocs de méthodes mais ne dit pas non plus comment les stocker.

Les différents types de composants sont en général stockés dans la même base. Par exemple, Van Slooten propose de stocker les cartes de routes dans la même base que les fragments de méthodes. De la même manière les cinq types de fragments (voir la section précédente) de l'approche *Fragments de méthodes* sont également stockés dans la même base de méthodes. Le même principe est appliqué dans l'approche *Composants contextuels* ainsi que dans l'approche proposée dans ce mémoire.

L'approche *Composants contextuels* ainsi que la notre divise la base de méthodes en deux niveaux : le premier est appelé *connaissance méthode* et le deuxième *méta-connaissance*. Le niveau *connaissance méthode* comporte la connaissance qui est effectivement réutilisable, c'est-à-dire les composants eux-mêmes tandis que le niveau de *méta-connaissance* comporte l'information nécessaire à la sélection et la réutilisation des composants. Dans le domaine de la réutilisation ces deux niveaux sont appelés la *connaissance réutilisable* et la *connaissance pour la réutilisation*.

Pour représenter la méta-connaissance l'approche *Composants contextuels* ainsi que la notre utilise la notion de *descripteur* qui permet de personnaliser les composants. Ceci facilite l'accès aux composants ainsi que leur sélection dans la base de méthodes. Chaque composant de méthode a un descripteur qui décrit son contexte de réutilisation. Un descripteur relie la situation dans laquelle le composant est pertinent à l'intention qu'il permet de satisfaire dans le processus d'ingénierie de systèmes. La situation de l'approche *Composants contextuels* détermine des caractéristiques qu'un projet doit avoir pour que le composant soit applicable dans son développement. Ces caractéristiques sont basées sur les résultats obtenus dans le projet EUROMETHOD [Frankson 94]. La situation dans notre approche fait référence aux domaines dans lesquels le composant est applicable et les activités de conception dans la réalisation desquelles le composant peut participer. L'intention dans les deux approches représente une intention d'ingénierie de systèmes qui peut être réalisée en appliquant le composant.

Une nouvelle manière de présenter les composants de méthodes pour qu'ils soient accessibles à un public plus large est de les offrir sous forme de bibliothèques électroniques accessibles par Internet. Ceci à été proposé dans [Ralyte 99b] et [Brinkkemper 00]. Brinkkemper souligne que la technologie Internet apporte des nouvelles perspectives dans la création et l'utilisation des méthodes. Tout d'abord, la disponibilité des méthodes sur Internet garantit à tout ingénieur d'application un accès facile à ces méthodes. L'évaluation des méthodes est plus rapide car grâce à Internet les utilisateurs de méthodes peuvent envoyer leurs appréciations, remarques et suggestions aux créateurs de méthodes ainsi que les futurs utilisateurs. Les méthodes peuvent utiliser différentes technologies multi-média pour les formations sur l'application de la méthode en utilisant des diapositives, des vidéos, des animations etc. La présentation hyper-texte des méthodes est plus attractive que la description sur le papier. On peut accéder rapidement à des parties de la description qui nous intéressent au moment donné.

3.4.3 Guidage dans la sélection et l'assemblage de composants

Le troisième problème que la stratégie de construction de méthodes par assemblage tente de résoudre, est l'existence d'un processus guidé qui permet de sélectionner des composants répondant aux besoins du projet en cours et d'assembler ces derniers afin d'obtenir une méthode adaptée à la situation de ce projet.

Comme nous avons montré dans les sections précédentes, plusieurs modèles ont été proposés pour définir les composants de méthodes. Malgré cela, le processus concernant leur sélection et leur assemblage reste un domaine de recherches peu évolué. Les approches comme *Composants contextuels* et *Blocs de méthodes* se limitent à la définition et à la représentation des composants. D'autres approches comme *Fragments de méthodes*, *MEMA*, l'approche proposée par Song ou celle de Van Slooten ne proposent pour le moment que des processus très génériques qui n'offrent aucun guidage à l'ingénieur de méthodes concernant la sélection et l'assemblage des composants.

Dans les approches *Fragments de méthodes*, MEMA ou celle de Van Slooten la première étape dans la construction d'une méthode situationnelle concerne la caractérisation du projet en cours. Ceci est basé sur la définition des facteurs de contingence qui peuvent être identifiés à l'aide des interviews, des questionnaires et d'autres techniques d'acquisition de la connaissance. MEMA, par exemple, utilise un cadre de référence défini pour ce propos. [Van Slooten 96] définit une liste de facteurs de contingence comme l'importance du projet, son impact sur l'organisation existante, la pression du temps, l'expérience de l'équipe de développement, la taille, les relations avec d'autres systèmes, la dépendance des autres projets, la clarté, la stabilité, la complexité, l'innovation etc.

Selon les auteurs de ces approches, les facteurs de contingence sont utilisés ensuite pour sélectionner les composants appropriés. L'approche MEMA par exemple, a un processus qui permet de déterminer la correspondance entre la caractérisation du projet à l'aide de leur cadre de référence et les descriptions des composants stockés dans une base de méthodes. Cependant, les autres auteurs restent assez flous dans leurs explications concernant l'évaluation de l'ensemble de facteurs de contingence et leur impact sur le choix d'un composant de méthode plutôt qu'un autre.

Le processus d'assemblage des composants sélectionnés reste également assez flou dans les approches MEMA et celle de Van Slooten. Par contre, l'approche *Fragments de méthodes* propose un processus assez détaillé guidant l'assemblage des fragments de méthodes sélectionnés au préalable. Ce processus consiste à assembler les fragments de produits et les fragments de processus correspondants. L'assemblage de deux fragments de produit consiste à identifier un ou plusieurs liens entre différents concepts des deux fragments. Des nouveaux concepts peuvent être créés pour permettre la connexion entre les fragments de produit. En ce qui concerne l'assemblage des fragments de processus, des liens de précédence entre les différentes activités des composants doivent être établis. Des contraintes d'assemblage sous forme des règles formalisées sont proposées pour assurer la cohérence et la complétude de l'assemblage obtenu. Cependant, cette approche se limite à l'assemblage des fragments complémentaires qui n'ont pas d'éléments communs.

Dans ce mémoire nous proposons un processus d'assemblage qui prend également en compte l'assemblage des composants qui se recouvrent partiellement, par exemple des composants qui

permettent de satisfaire le même objectif mais qui proposent des solutions différentes. Nous donnons un nom d'*intégration* à cette technique. L'intégration de tels composants permet d'obtenir un nouveau composant encore plus riche que les deux initiaux. De plus, notre processus d'assemblage est fondé sur des opérateurs formalisés et offre également des règles de validation de la cohérence de l'application de ces opérateurs et de la complétude du résultat obtenu.

Song dans son approche parle d'adaptation et d'amélioration des méthodes existantes plutôt que de la construction d'une nouvelle méthode. Il propose deux types d'assemblage qu'il appelle *basé fonction* et *basé qualité*. Le premier type permet d'ajouter des nouvelles fonctionnalités dans une méthode tandis que le deuxième permet d'améliorer la qualité de la méthode, d'augmenter son efficacité en lui ajoutant de nouvelles propriétés. Les deux types d'assemblage concernent les deux niveaux de composants (voir la section 3.4.1.3). Les processus d'assemblage restent cependant informels, proposant uniquement des raisons, des heuristiques, des exemples mais pas de guidage systématique.

Notre approche prend également en compte ces deux objectifs d'assemblage en proposant un processus guide à chaque fois.

3.5 Utilisation d'un outil CAME

Une croissance importante du nombre de méthodes d'ingénierie de systèmes et de leurs environnements du support a influencé l'apparition d'une nouvelle technique d'ingénierie de méthodes appelée Atelier d'Ingénierie de Méthodes (AIM) (ou CAME - Computer Aided Method Engineering) [Slooten 93], [Kumar 92]. L'ingénierie de méthodes dans ce domaine est définie selon Heym et Osterle [Heym 93] comme un processus discipliné pour construire, évaluer ou modifier une méthode par le moyen de spécification des composants de méthode et des relations entre eux. Si les ateliers de génie logiciel (AGL) sont des outils qui aident dans le développement de systèmes d'information, les outils CAME visent à aider dans le développement de méthodes. L'objectif d'un outil CAME est d'aider l'ingénieur de méthodes à construire de nouvelles méthodes et à modifier des méthodes existantes.

Selon Harmsen [Harmsen 94], un outil CAME doit offrir les fonctionnalités suivantes :

- *La définition et l'évaluation de facteurs de contingence*. Pour pouvoir choisir des composants de méthodes correspondants aux exigences du projet en cours, des règles et des facteurs de sélection propres à ces composants doivent être définis par l'ingénieur de méthodes. Pour un profil de projet donné et une base de méthodes, l'outil CAME sélectionne et assemble une méthode appropriée.

- *Le stockage de composants de méthodes*. Afin de pouvoir manipuler les composants de méthodes avec un outil CAME, il est nécessaire de les stocker dans une base de méthodes. Les nouvelles méthodes peuvent être ajoutées dans la base, les composants peuvent être modifiés ou éliminés de la base.

- *L'extraction et l'assemblage des composants*. L'ingénieur de méthodes doit avoir la possibilité de sélectionner des composants dans la base de méthodes. Un langage d'interrogation pour accéder au

contenu de la base a besoin d'être défini. La connaissance permettant le développement d'une nouvelle méthode doit être disponible.

- *La validation et la vérification de la méthode construite*. L'outil CAME devrait non seulement aider à réaliser les tâches d'assemblage et de sélection mais aussi à vérifier la méthode résultante. L'outil devrait donc incorporer des directives permettant d'assurer l'exactitude de la méthode.

- *L'adaptation de la méthode obtenue*. L'outil CAME devrait offrir la fonctionnalité de l'adaptation dynamique de la méthode. Il doit également permettre de compléter la base de méthodes compte tenu de l'expérience acquise.

L'environnement d'un outil CAME doit être composé de deux sous-environnements: celui de l'ingénierie d'applications et celui de l'ingénierie de méthodes. Le lien entre les deux sous-environnements peut être établi grâce au concept de la *base de méthodes* qui est utilisée par les deux. L'environnement d'ingénierie d'applications doit permettre d'exécuter les fonctionnalités de l'outil CAME comme la définition et l'évaluation des facteurs de contingence, la sélection des composants dans la base de méthodes, l'assemblage de composants et l'évaluation de la méthode obtenue. L'environnement d'ingénierie de méthodes doit permettre de définir et d'améliorer les composants de méthodes.

Même si aujourd'hui les fonctionnalités qu'un l'outil CAME devrait fournir sont bien définies, des travaux considérables doivent encore être réalisés pour implémenter ces fonctionnalités. Cependant, un nombre de produits meta-CASE et de prototypes ont été développés en réalisant partiellement certaines fonctionnalités. Nous pouvons citer pour exemple, Maestro II [Merbeth 91], MEET [Heym 93] MetaEdit+ [Kelly 96], Decamerone [Harmsen 95], Mentor [Si-Said 96] et MERU [Prakash 99], [Prakash 98]. Decamerone est dans un état préliminaire de développement mais permet l'assemblage de fragments de processus et de produit qui ont été sélectionnés à l'aide des facteurs de contingence du projet. Cependant, il ne fournit pas de guidage ni pour l'ingénierie des méthodes ni pour l'ingénierie d'applications. L'outil MEET propose un modèle de représentation de méthodes qui peut être utilisé pour standardiser, comparer et intégrer des différentes méthodes. MetaEdit+ inclut un nombre d'instanciations principalement sur les aspects produit d'une vingtaine de méthodes. Maestro II est un outil méta-CASE sur lequel Decamerone a été développé. L'accent, dans l'outil Mentor, a été mis sur l'unification des aspects de produit et de processus des méthodes ainsi que sur le guidage et le support qui peut être fourni pour l'ingénieur de méthodes et l'ingénieur d'applications. MERU consiste en deux parties : la première aide à formuler les exigences sur la méthode à construire, la deuxième génère la méthode. Les exigences sur la méthode sont exprimées en utilisant un méta-modèle spécifique appelé Method View. La génération de la méthode utilise la technique d'assemblage. La notion de composant de méthode est définie pour permettre ceci.

L'utilisation de la méta-modélisation associée à un environnement CAME dans l'ingénierie des méthodes permet de réaliser deux objectifs simultanément : premièrement nous pouvons comparer les méthodes de manière analytique, deuxièmement nous pouvons placer les méthodes dans un environnement où elles ont une plate-forme qui permet de les stocker et de les représenter. Les premiers travaux dans ce domaine ont été concentrés sur la définition des langages de méta-

modélisation [Brinkkemper 90], [Tolvanen 93] ou sur la construction des environnements pour eux [Chen 91], [Sorenson 88], [Smolander 91b].

3.6 Utilisation des patrons génériques

Le concept de *patron* est né grâce au travail de Alexander et al. qui a proposé d'utiliser celui-ci dans le domaine de l'architecture des bâtiments [Alexander 79]. Puis, ce concept a été tout d'abord emprunté dans le domaine du développement de logiciels et surtout dans celui des approches orientées-objet. Il a été appliqué dans la programmation de logiciels [Beck 97], [Bushmann 96], la conception de systèmes [Coad 92], [Coad 96], [Gamma 94], [Coplien 95], [Vlissides 96], [Front 97], [Rieu 99] la modélisation des données [Hey 96] et l'analyse de systèmes [Fowler 97].

Alexander définit un patron comme *"une solution à un problème qui se produit souvent dans notre environnement décrite d'une telle manière que l'on peut utiliser cette solution un million de fois sans pour autant faire la même chose deux fois"*. L'essence d'un patron est dans le fait qu'il propose une solution réutilisable dans toute situation où le problème concerné par le patron apparaît.

Grâce à son pouvoir de réutilisation, ce concept a été introduit récemment dans le domaine de l'ingénierie de méthodes [Rolland 96a], [Rolland 96b], [Rolland 00a], [Rolland 00b], [Denéckère 01].

Le travail proposé dans [Rolland 96a]et [Rolland 96b] introduit la notion de patron comme un moyen pour modéliser le comportement commun dans la construction des méthodes. [Rolland 96a] définit plusieurs patrons de construction référencés par les verbes d'intentions qu'ils permettent de satisfaire dans le processus de construction d'une méthode, comme par exemple *Identifier*, *Décrire*, *Construire*, *Définir*, *Valider* et *Affiner*. Ces patrons peuvent être appliqués dans la construction de plusieurs méthodes. Ces patrons sont génériques dans le sens où ils peuvent être utilisés par un ingénieur de méthodes dans le processus de construction de plusieurs méthodes.

Le projet ELEKTRA propose un langage à base de patrons, appelé *patrons de décision*, pour décrire des meilleures expériences dans la gestion de changements dans des systèmes gérant des processus d'organisation des entreprises. Une méthode pour définir les patrons est également définie permettant d'organiser et d'affiner les patrons existants et de découvrir de nouveaux patrons [Rolland 00a], [Rolland 00b].

[Denéckère 01] propose une approche d'extension de méthodes existantes en utilisant des patrons génériques. De plus, cette approche comporte une technique d'organisation de ces patrons avec un guidage pour leur réutilisation ainsi qu'une technique de conception de nouveaux patrons d'extension à l'aide de méta-patrons.

3.7 Evaluation des méthodes

L'étape de la validation d'une nouvelle méthode est très importante surtout si la méthode est sensée être générique et indépendante d'un projet particulier. Plusieurs techniques de validation des méthodes

ont été proposées à ce jour. Pratiquement chaque nouvelle méthode, chaque nouvelle approche passe par l'étape de validation avant d'être diffusée. Le plus souvent, les méthodes sont évaluées en les appliquant pour modéliser des cas appartenant aux différents domaines d'application.

Une autre technique très répandue est basée sur l'utilisation des groupes de personnes qui ne connaissent pas la méthode et qui sont invitées à l'appliquer sur un cas, en général le même pour tout le monde. Un questionnaire suit le plus souvent la séance du travail pour évaluer la clarté de la méthode, son adaptabilité au problème, sa facilité d'utilisation, le temps nécessaire pour assimiler les notions utilisées etc.

Si la méthode a un support outillé, elle peut être évaluée en expérimentant les capacités de cet outil. Ceci peut être fait par les études des cas individuels ou groupés en appliquant l'outil correspondant. Les remarques des testeurs sont prises en compte dans la nouvelle version de l'outil. Mais aussi, elles peuvent faire apparaître des problèmes méthodologiques qui doivent être résolus avant la modification de l'outil.

Toute utilisation de la méthode sur un cas réel est également une sorte d'évaluation de celle-ci. Tout ingénieur d'application est invité à donner ses appréciations après avoir utilisé une méthode dans la conception d'un système réel. Son expérience, ses remarques et ses propositions sont cruciales pour l'évolution de la méthode.

4. CONCLUSION

Dans ce chapitre nous avons considéré la discipline d'ingénierie des méthodes comme une méthode elle-même. C'est une méta-méthode pour construire des méthodes d'ingénierie de systèmes. Nous avons étudié sa perspective processus en modélisant tout d'abord celle-ci par un modèle de processus stratégique.

Ce modèle nous a permis de caractériser les différentes approches d'ingénierie des méthodes à l'aide des intentions identifiées dans ce domaine et les stratégies pour satisfaire ces intentions.

En parcourant les stratégies de ce modèle nous avons identifié les approches qui nous semblaient suivre ces stratégies et nous les avons comparées les unes aux autres.

Ce parcours a permis de constater que le fondement de la plupart des techniques de construction, d'évaluation et de comparaison de méthodes est basé sur la méta-modélisation. De plus, les techniques de construction de méthodes rigides sont pratiquement remplacées par des techniques de construction dynamique à base des composants de méthodes et proposant souvent un support outillé. Ces techniques permettent en général de construire des méthodes spécifiques aux projets de telle manière que, à son tour, elles peuvent être réutilisées dans la construction de nouvelles méthodes.

PARTIE I

RE-INGENIERIE DES METHODES A BASE DE COMPOSANTS

Dans la première partie de ce mémoire, nous proposons une approche d'ingénierie des méthodes modulaires avec l'objectif de réutiliser ensuite les différents fragments des telles méthodes dans la construction d'autres méthodes.

La Figure 5 résume notre approche d'ingénierie des méthodes à base de composants. Un des éléments principaux de cette approche est le méta-modèle de méthodes qui perçoit toute méthode comme un assemblage de composants réutilisables que nous appelons des *composants de méthode*. Tous les composants de méthode sont des unités de modélisation indépendante. Ils peuvent être stockés dans une base de méthodes indépendamment les uns des autres et réutilisés par la suite dans la construction d'autres méthodes.

Le deuxième élément principal de cette approche est le processus de ré-ingénierie des méthodes existantes. Dans ce travail nous proposons un modèle de processus de ré-ingénierie qui, en respectant notre méta-modèle, aide l'ingénieur à transformer toute méthode en un assemblage de composants. Ces derniers sont stockés dans une base de méthodes et peuvent être réutilisés ultérieurement.

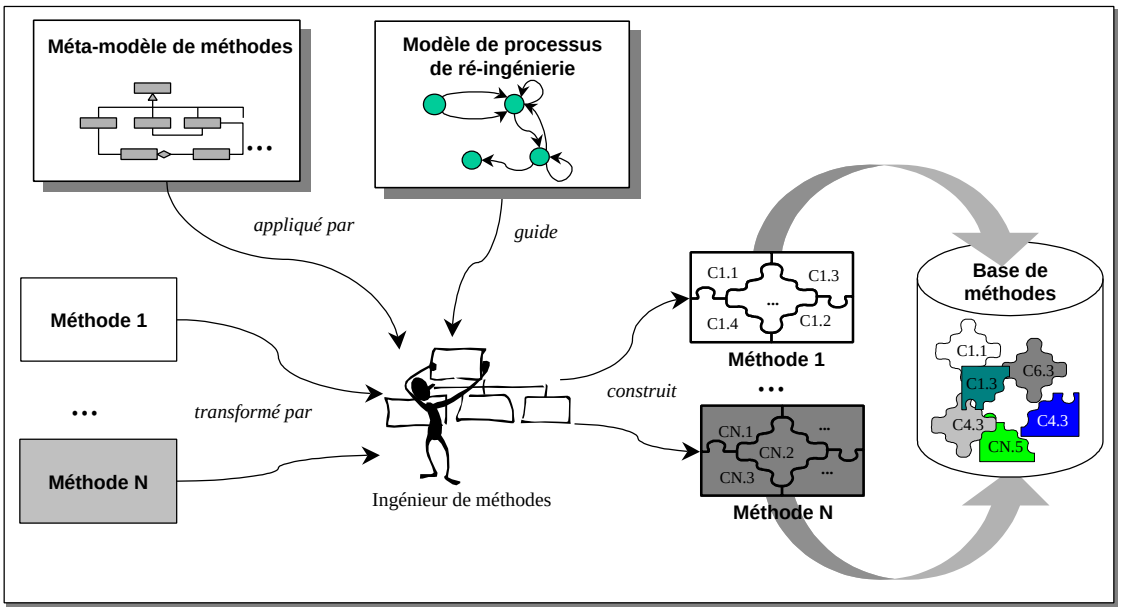


Figure 5 : Modélisation des méthodes à base de composants réutilisables

Cette partie du mémoire est ordonnancée en deux chapitres : le Chapitre 3 présente le méta-modèle des méthodes modulaires et le Chapitre 4 définit le processus de ré-ingénierie des méthodes existantes.

CHAPITRE 3

Méta-modèle de méthodes à base de composants

Ce chapitre est consacré à la présentation du méta-modèle de méthodes proposé dans ce mémoire. Ce méta-modèle est utilisé pour représenter les méthodes sous forme de modules réutilisables.

Nous avons adopté cette approche modulaire de représentation des méthodes de développement des systèmes logiciels pour faciliter la construction des méthodes situationnelles. Suivant notre approche, une méthode est vue comme un ensemble de modules que nous appelons des *composants de méthode*. La structure de composant proposée dans ce chapitre permet de stocker les composants de méthodes indépendamment de leurs méthodes d'origine, dans une base méthodologique et de les réutiliser dans la construction d'autres méthodes. Ainsi, ils deviennent des modules permettant la construction de nouvelles méthodes. Ils peuvent aussi être utilisés pour enrichir des méthodes existantes. La taille des composants varie ; un composant peut représenter une activité particulière d'une méthode ou être associé à une méthode entière.

Dans ce chapitre, nous proposons une approche pour représenter une méthode sous forme d'un ensemble de composants réutilisables. La section 1 du chapitre décrit le fondement de cette approche. La section 2 propose un modèle de représentation de méthodes à base de composants. La section 3 développe la notion de composant de méthode tandis que la section 4 va encore plus loin dans les détails concernant la partie principale du composant, celle qui représente le processus d'ingénierie capturé dans ce dernier. Enfin, la section 5 conclut la proposition de notre méta-modèle.

1. VUE GENERALE D'UNE METHODE

Selon notre point de vue – partagé par plusieurs auteurs ([Kronlof 93], [Smolander 91], [Wynekoop 93], [Lyytinen 89], [Prakash 94], [Brinkkemper 90], [Seligmann 89], [Harmsen 94], [Brinkkemper 96]) – une méthode comporte un ou plusieurs *modèles de produit* et un ou plusieurs *modèles de processus*.

La Figure 6 montre la vue générale d'une méthode. Chaque méthode est composée de deux parties : une partie produit et une partie processus. La partie produit est représentée par un ou plusieurs modèles de produit et la partie processus par un ou plusieurs modèles de processus. Chaque modèle de produit est couplé à un modèle de processus correspondant. Le modèle de produit représente la structure d'une classe de produits obtenus en appliquant la méthode dans différentes applications tandis que le modèle de processus représente la démarche à suivre pour obtenir le produit cible.

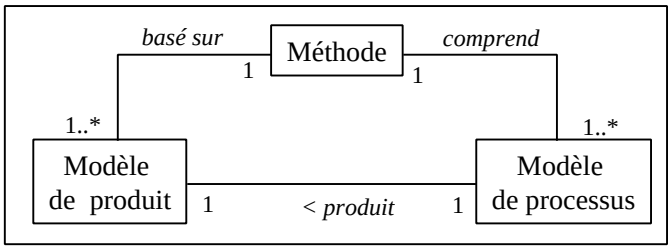


Figure 6 : Méta-modèle de méthodes

La méthode OOSE [Jacobson 92] par exemple, propose deux modèles pour l'étape d'analyse de développement des systèmes: *"le modèle des cas d'utilisation"* et *"le modèle d'analyse"*. La méthode *OMT* [Rumbaugh 91] est composé de trois modèles : *"le modèle objet"*, *"le modèle dynamique"*, *"le modèle fonctionnel"*. Chacun de ces modèles définit la structure du produit qui est obtenu en appliquant le modèle et une démarche prescrivant la construction du produit correspondant.

L'objectif de notre travail est de représenter les méthodes sous forme d'un ensemble de *modules*. Chaque module est une partie de processus couplé aux parties de produit utilisées par ce processus. De plus, ce module de processus devrait être réutilisable indépendamment de sa méthode origine pour la construction d'autres méthodes. Nous appelons ces modules des *composants de méthode* ou, tout simplement, des *composants*.

Prenons de nouveau comme exemple la méthode OOSE [Jacobson 92]. Comme nous l'avons dit plus haut, cette méthode propose deux démarches complémentaires pour l'étape d'analyse dans le développement d'un système d'information. La première permet de définir certaines fonctionnalités d'un système qui sont décrites sous forme de cas d'utilisation. La deuxième permet de définir la structure d'un système sous forme d'un diagramme de classes. Chacune de ces deux démarches peut être réutilisée indépendamment de la méthode d'origine et, par conséquent, être représentée comme un composant de méthode réutilisable en lui associant les parties de produit correspondantes.

Chacune de ces démarches peut également être décomposée en directives plus fines et ainsi de suite. Par exemple, la démarche de *"définition des cas d'utilisation"* peut être décomposée en directives de

"découverte des cas d'utilisation" et directives *"d'écriture des cas d'utilisation"*. Chacune de ses directives peut aussi devenir un composant de méthode.

Chaque composant peut être réutilisé dans la construction des nouvelles méthodes ou pour l'enrichissement des méthodes existantes.

2. REPRESENTATION MODULAIRE D'UNE METHODE

2.1 Notion d'un composant de méthode

Dans ce travail, nous proposons de voir une méthode comme un ensemble de modules que nous appelons des composants de méthode. Cette vue est favorable à l'adaptation et à l'extension des méthodes ainsi qu'à la construction de nouvelles méthodes en fonction des spécificités du projet en cours, ce qui est le but de l'ingénierie des méthodes situationnelles.

La Figure 7 représente le méta-modèle de méthodes que nous proposons dans ce mémoire en utilisant la notation *UML* [UML 2000]. Selon ce méta-modèle, une méthode est vue comme un ensemble de composants de différents niveaux de granularité, la méthode elle même étant un composant de plus haut niveau.

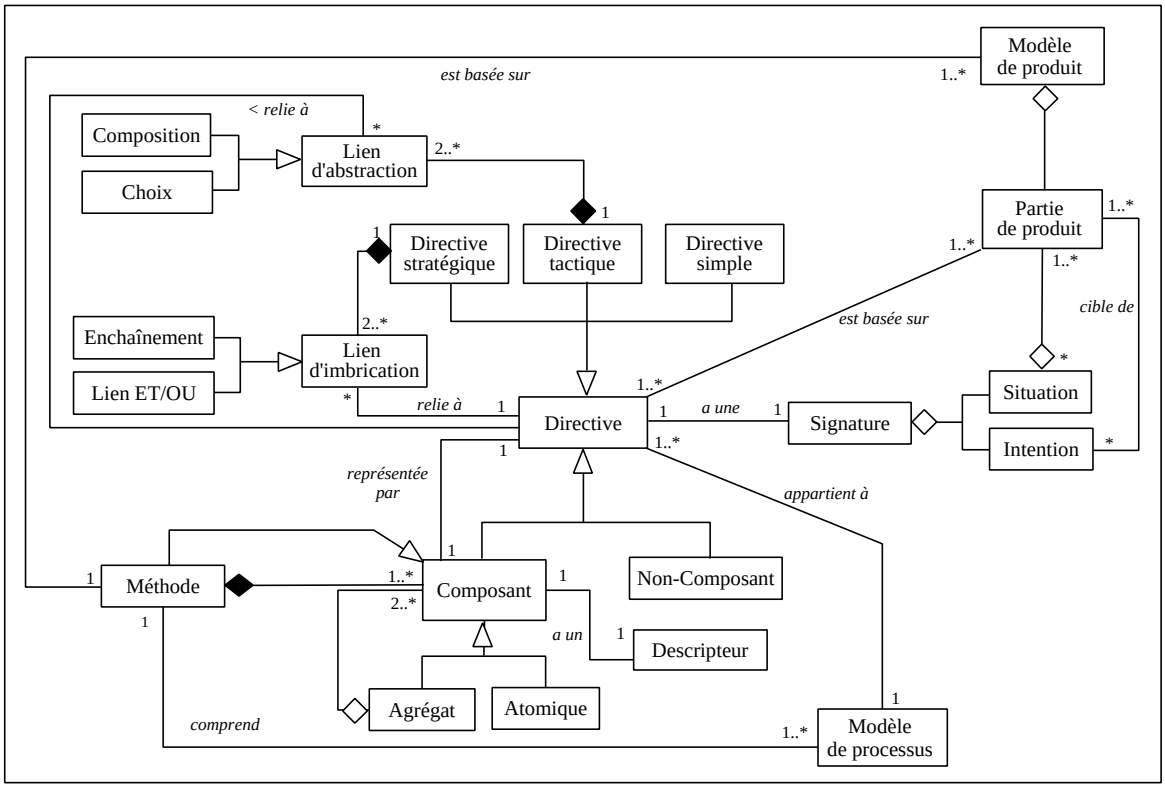


Figure 7: Méta-modèle des méthodes modulaires

Notre définition d'un composant de méthode est dirigée par la décomposition d'une démarche de méthode en sous-démarches que l'on appelle des *directives*. Un composant de méthode est vu comme un fragment de processus associé à un fragment de produit qui garantit le bon déroulement de ce processus. La modélisation retenue ici est donc "centrée processus" ; la partie principale d'un composant étant la partie processus. Mais celle-ci n'a de sens qu'associée à sa partie de produit.

Suivant ce raisonnement, la décomposition d'une méthode en composants est basée sur la décomposition de son modèle de processus en directives. Un composant de méthode est vu comme une *directive* couplée aux *parties de produit* nécessaires à l'exécution de la démarche capturée dans cette directive (Figure 7).

La partie produit du composant est en quelque sorte un sous-modèle de l'un des modèles de produit de la méthode correspondante. Plus précisément, cette partie contient les différentes parties de produit appartenant au modèle de produit de la méthode, nécessaires à l'application du processus capturé dans la partie processus du composant. Nous développons le concept de partie de produit à la section 3.3.

De la même manière, la partie processus du composant est un sous-modèle d'un des modèles de processus de la méthode correspondante. Ce sous-modèle de processus est représenté sous forme d'une directive. C'est la partie la plus importante d'un composant car elle définit une démarche que l'ingénieur de développement doit suivre pour atteindre l'objectif du composant. Le concept de la directive est introduit à la section 3.1 et présenté en détail à la section 4 de ce chapitre.

Un autre concept important dans le méta-modèle de méthodes modulaires (Figure 7) est la *signature* de la directive. Toute directive a une signature qui représente les conditions d'utilisation de celle-ci ainsi qu'un corps comportant le processus d'ingénierie. Un composant étant fait d'une directive, la signature de cette dernière est aussi la signature du composant. La notion de la signature est développée à la section 3.2 de ce chapitre.

Finalement, le concept de *descripteur* (Figure 7) est associé à chaque composant et permet de définir le contexte de réutilisation de celui-ci. La section 3.4 de ce chapitre décrit plus en détail la notion de descripteur.

Prenons comme exemple un composant issu de la méthode CREWS-*L'Ecritoire* [Rolland 98b], [Rolland 98e], [Tawbi 99b], dont l'objectif est de guider l'ingénieur d'applications dans l'écriture de scénarios en prose libre. La méthode CREWS-*L'Ecritoire* est destinée à la découverte des besoins d'un système d'information sous forme de buts. La démarche qu'elle propose est basée sur l'écriture de scénarios et la découverte de buts à partir de ceux-ci. La partie concernant l'écriture des scénario peut être vue comme un composant de méthode réutilisable dans la construction d'autres méthodes. Ce composant définit le concept d'un scénario décrivant le comportement d'un système d'information dans la réalisation d'une de ses fonctionnalités. La directive de ce composant décrit la démarche que l'ingénieur d'applications doit suivre pour aboutir au résultat souhaité. Un certain nombre de parties de produit issues de la même méthode sont associées à cette démarche. Ce sont les parties de produit comme *"agent"*, *"action"*, *"but"* et *"scénario"*, nécessaires au bon déroulement de la démarche. La directive du composant décrit alors comment écrire un scénario complet et cohérent, tandis que la partie produit définit la structure que le scénario obtenu doit avoir.

2.2 Typologie des directives

Etant donné que le concept de *directive* représente la partie principale du composant, ce chapitre est consacré à la définition de cette notion. Dans cette section nous introduisons les typologies de directives et les liens entre celles-ci.

Nous distinguons deux typologies de directives. La première classe les directives selon leur contenu. Suivant la forme de leur représentation, leur complexité et leur richesse, elles peuvent être *simples*, *tactiques* ou *stratégiques* (Figure 7).

La seconde classe les directives selon leur taille et leur importance dans le souci de la réutilisation, c'est-à-dire qu'elle définit si une directive est un composant de méthode ou tout simplement une partie de la directive d'un composant (Figure 7).

Les deux typologies sont orthogonales l'une à l'autre. Nous les décrivons dans les deux sous-sections suivantes et nous démontrons en même temps la richesse de la structure des directives.

2.2.1 Simple, tactique ou stratégique

Toutes les directives ne sont pas décrites de la même manière. Suivant la méthode, elles sont plus ou moins riches et peuvent être décrites d'une manière plus ou moins formelle.

D'un côté la taille des directives varie d'une action atomique jusqu'à une démarche complète de la méthode. D'un autre côté, la représentation des directives varie de la description textuelle jusqu'à la représentation par un graphe complexe de processus.

Une directive *simple* est une directive sans structure, elle peut être associée à une description textuelle ou à une action plus ou moins complexe à exécuter.

Une directive *tactique* est une directive complexe utilisant une *structure d'arbre* pour relier ses sous-directives. Comme le montre la Figure 7, elle est reliée à travers un *lien d'abstraction* à un ensemble de directives qui sont de niveau d'abstraction plus bas. Chaque élément de cette directive appartient à l'un des trois types de directive, c'est-à-dire simple, tactique ou stratégique.

Une directive *stratégique* est une directive complexe utilisant une *structure de graphe* pour relier ses sous-directives. C'est une directive multi-démarche, car elle propose plusieurs démarches possibles pour satisfaire son objectif. Elle relie à travers un *lien d'imbrication* (Figure 7) plusieurs directives de même niveau d'abstraction. Comme dans le cas de la directive tactique, chaque élément de la directive stratégique appartient à l'un des trois types de directive.

Dans les sous-sections suivantes nous introduisons les liens entre les directives pour démontrer la variété et la richesse des directives que nous proposons dans notre méta-modèle.

La section 4 montre de façon plus détaillée le contenu et la forme des directives.

2.2.1.1 Lien d'abstraction

Le lien d'abstraction est utilisé pour exprimer la structure des directives tactiques. Il y a deux types de lien d'abstraction: *la composition* et *le choix*. Les deux liens construisent une structure d'arbre en utilisant les connecteurs ET et OU respectivement. Leurs représentations graphiques simplifiées sont illustrées à la Figure 8.

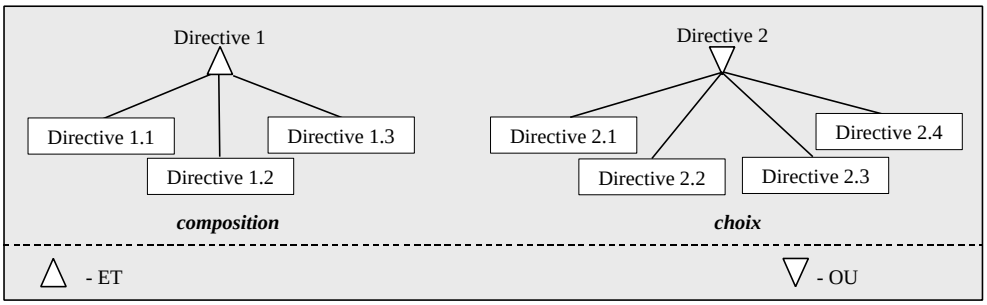


Figure 8 : Représentations graphiques des liens de composition et de choix.

Pour illustrer ces liens ainsi que d'autre liens entre les directives nous représentons pour le moment une directive comme une boîte noire sans préciser le contenu de cette boîte et en montrant seulement son objectif.

2.2.1.1.1 Composition

Un lien de *composition* unit un ensemble de directives dans une directive tactique sous forme d'un plan définissant l'ordre dans lequel les sous-directives doivent être exécutées. L'exécution d'une directive tactique de ce type consiste à exécuter toutes ses sous-directives dans un ordre prédéfini par le lien de composition.

Prenons comme exemple une directive de la méthode CREWS-*L'Ecritoire* qui permet, à partir d'un but du système d'information à développer et identifié au préalable, de découvrir des buts alternatifs en appliquant le modèle de but. Cette directive est illustrée à la Figure 9. Elle est composée de quatre sous-directives. Chacune d'entre elles doit être réalisée une ou même plusieurs fois pour satisfaire l'objectif de la directive tactique. Tout d'abord, le but initial doit être réécrit suivant le modèle de but (1), puis plusieurs valeurs alternatives doivent être proposées à tous les paramètres du but (2), ce qui veut dire que la deuxième directive doit être répétée plusieurs fois. Ensuite, un ou plusieurs nouveaux buts peuvent être construits (3) et finalement l'un d'entre eux doit être sélectionné (4). Le plan définissant l'ordre de la réalisation de ces directives est inclus dans la directive tactique.

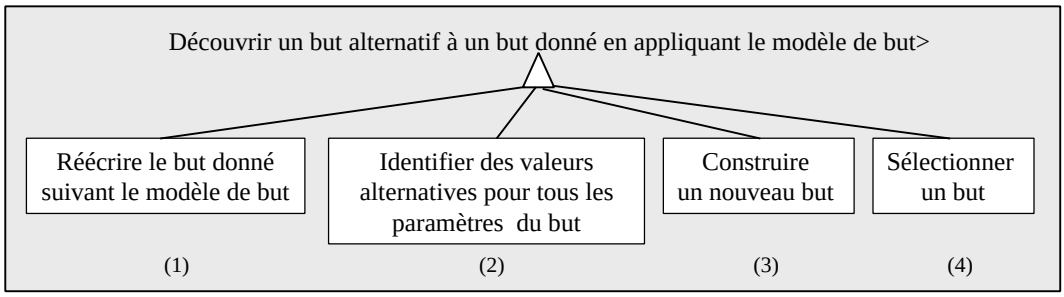


Figure 9 : Exemple de directive tactique à base de lien de composition

2.2.1.1.2 Choix

Un lien de choix relie un ensemble de directives dans une directive tactique sous forme d'un ensemble d'alternatives. Chaque sous-directive représente une manière différente pour satisfaire l'objectif de la directive tactique. En d'autres termes, l'objectif de la directive tactique est affiné par des objectifs plus précis proposant chacun une manière différente pour satisfaire l'objectif de départ. L'exécution d'une directive tactique de ce type consiste à choisir l'une de ses sous-directives, la plus adaptée à la situation donnée, et à l'exécuter. Elle définit également comment choisir une des sous-directives.

Par exemple, la méthode CREWS-*L'Ecritoire* propose quatre manières différentes d'écrire des scénarios en prose libre. Chaque manière est représentée sous forme d'une directive spécifique. Une abstraction de ces quatre directives est une directive tactique dont l'objectif est une abstraction des objectifs de celles-ci, c'est-à-dire d'aider l'ingénieur d'applications à écrire des scénarios. La Figure 10 illustre comment cette directive tactique relie toutes ses sous-directives en termes de choix satisfaisant le même objectif mais de manière différente. De plus, cette directive propose des arguments aidant l'ingénieur de développement à choisir l'une de ces manières selon son expérience d'écriture de scénarios. Il peut choisir un guidage relatif au style d'écriture du scénario (1), un guidage concernant le contenu du scénario (2) ou les deux (3). Un expert du domaine n'a pas besoin d'être guidé dans cette tâche ; c'est un choix possible (4).

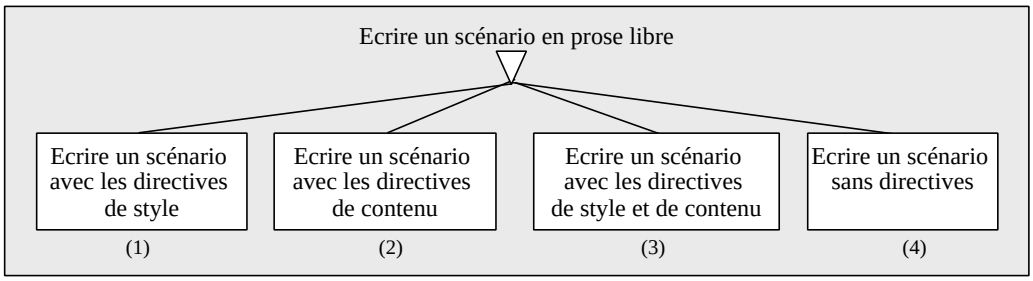


Figure 10 : Exemple de directive tactique à base de lien de choix

2.2.1.2 Lien d'imbrication

Le lien d'imbrication est utilisé pour exprimer la structure d'une directive stratégique. Il y a deux types de lien d'imbrication : *le lien ET/OU* et *l'enchaînement*. Les deux liens construisent une structure de graphe dont les nœuds sont des intentions de processus d'ingénierie et les liens entre ces

nœuds sont les directives permettant de satisfaire ces intentions. La directive stratégique est alors un graphe de directives reliées par des liens ET/OU et l'enchaînement ayant une intention d'entrée (Démarrer) et une intention de sortie (Arrêter). La représentation graphique simplifiée de la directive stratégique est illustrée à la Figure 11.

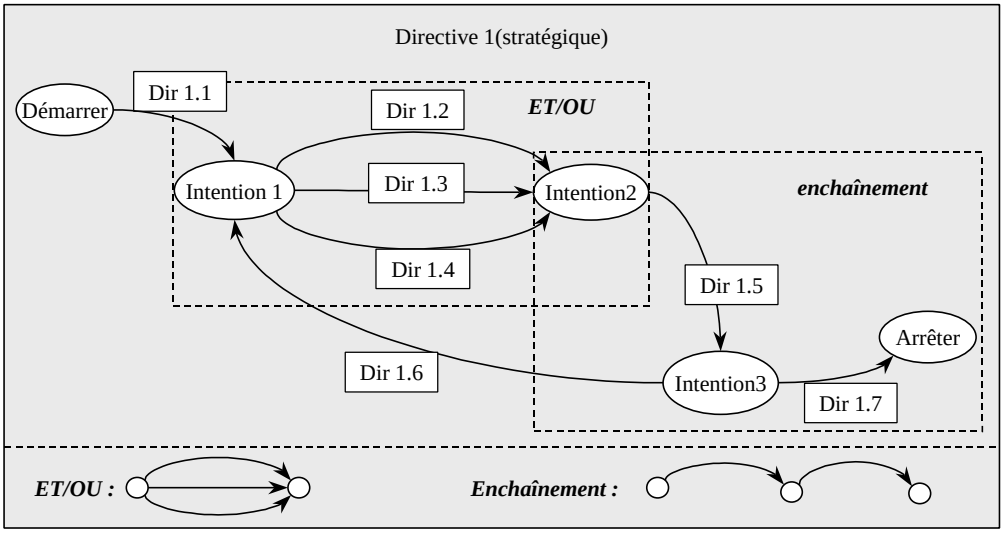


Figure 11 : Représentation de la directive stratégique et des liens ET/OU et l'enchaînement.

2.2.1.2.1 Lien ET/OU

Un lien d'imbrication *ET/OU* regroupe un ensemble de sous-directives d'une directive stratégique ayant les même points de départ et d'arrivée dans le graphe de celle-ci dans une nouvelle directive. Toutes les sous-directives de cette dernière ont le même objectif (la même intention d'arrivée) mais les moyens ou les manières de le satisfaire divergent. Le produit obtenu en exécutant chaque directive n'est pas exactement le même, la différence est soit dans leur structure soit dans leur contenu. Le lien ET/OU est différent du lien choix car il relie les directives qui ne sont pas forcément exclusives ; elles peuvent être complémentaires. L'ingénieur d'applications est guidé dans la sélection des sous-directives. L'exécution d'une telle directive consiste à exécuter une ou plusieurs sous-directives suivant les besoins en cours.

Illustrons cette définition avec la méthode CREWS-*L'Ecritoire*, qui définit plusieurs chemins différents menant au même objectif – la découverte des buts du système. Prenons l'exemple suivant : à partir d'un but identifié au préalable et de son scénario associé (l'intention *"Ecrire un scénario"* à la Figure 12 a été déjà satisfaite), la méthode propose trois directives pour découvrir d'autres buts (satisfaire l'intention *"Découvrir un but"* à la Figure 12). L'ingénieur de développement peut découvrir des buts complémentaires (1) ou alternatifs (2) à celui de départ, ou bien des buts d'un niveau d'abstraction plus bas (3). Le lien ET/OU associe ces trois sous-directives de telle manière que l'ingénieur de développement puisse les choisir et les exécuter suivant ses besoins en cours. Il peut les appliquer toutes les trois l'une après l'autre dans n'importe quel ordre ou en choisir seulement une ou deux parmi les trois.

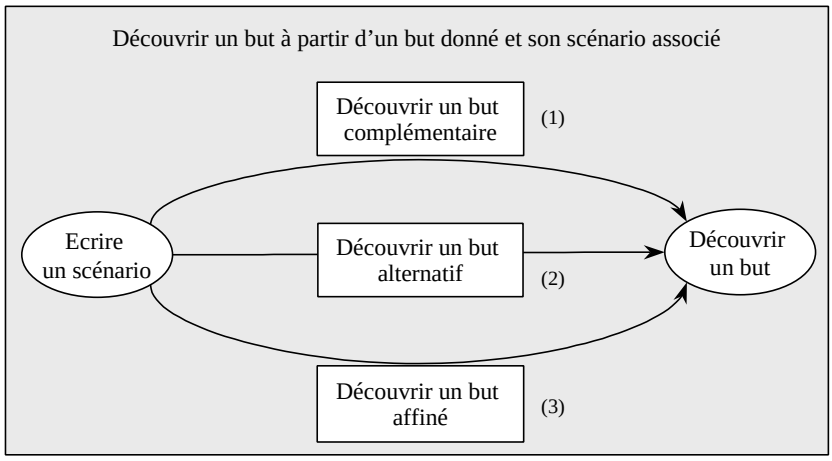


Figure 12 : Exemple de lien ET/OU dans une directive stratégique

De la même façon, chacune de ces sous-directives peut être une directive simple, tactique ou stratégique.

2.2.1.2.2 Enchaînement

Un lien *d'enchaînement* relie deux sous-directives d'une directive stratégique qui s'enchaînent l'une après l'autre dans son graphe en construisant ainsi une autre directive. Celle-ci exprime l'enchaînement possible de deux directives et, par conséquent, elle est de même niveau d'abstraction que ses sous-directives.

Prenons pour exemple l'enchaînement de deux sous-directives de la directive stratégique représentant la méthode CREWS-*L'Ecritoire* illustrée à la Figure 13. Cet enchaînement propose de faire suivre l'écriture d'un scénario par sa conceptualisation. La première sous-directive aide à écrire un scénario pour un but identifié au préalable (l'intention *"Découvrir un but"* à la Figure 13 a été déjà satisfaite). La deuxième sous-directive suit la première et aide à conceptualiser le scénario obtenu lors de l'exécution de la première directive. La directive qui propose d'écrire les scénarios en prose libre, a déjà été présentée dans la Figure 10. C'est une directive tactique.

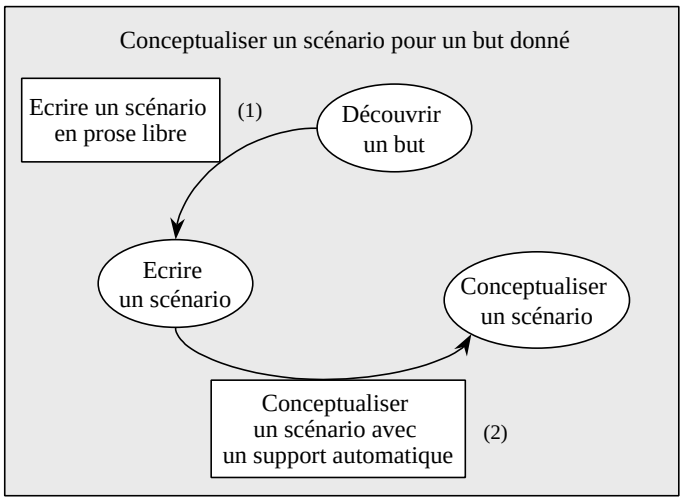


Figure 13 : Exemple de lien d'enchaînement dans une directive stratégique.

2.2.2 Composant versus non-composant

Suivant le méta-modèle présenté à la Figure 7, un composant de méthode est une directive de processus de développement d'un système. Les parties de produit associées à la directive définissent le produit cible de ce processus. En tant que directive, le composant a une signature qui définit son objectif et ses conditions d'application. Cependant, certaines directives ne sont pas des composants mais de simple directives.

Dans cette section nous introduisons la deuxième typologie qui détermine si une directive est représentée par un composant de méthode ou non.

Une directive pouvant être réutilisée indépendamment de sa méthode d'origine est définie en tant que composant de méthode en lui associant les parties de produit nécessaires à sa réalisation ainsi qu'un descripteur définissant les conditions de sa réutilisation (Figure 7). Par exemple, la directive que nous avons illustrée à la Figure 9 est un module de processus réutilisable. Par conséquent, on peut la définir en tant que composant de méthode.

Certaines directives ne sont pas assez importantes pour être réutilisées en tant que modules réutilisables dans la construction des méthodes. Elles font alors partie de directives qui, à leur tour, peuvent être représentées par des composants. Par exemple, les sous-directives de la directive tactique représentée à la Figure 9 ne sont pas réutilisables sans celle qui les intègre. Dans ce cas, elles ne sont pas définies par des composants de méthode. La directive de la Figure 10 au contraire, est définie en tant que composant de méthode ainsi que toutes ses sous-directives.

2.3 Niveaux de granularité des composants

Comme nous l'avons vu à la section précédente, les directives sont définies à différents niveaux d'abstractions et elles existent à différents niveaux de granularités. Puisque la base d'un composant est sa directive, les composants également existent à différents niveaux de granularité. La taille d'un composant varie d'une directive simple jusqu'à la représentation de la démarche complète d'une méthode. Une méthode peut alors être faite de composants à différents niveaux de granularité.

Comme le montre la Figure 7, il y a deux types de composants : des composants *atomiques* et des composants *agrégats*. Un composant agrégat est composé de plusieurs composants atomiques et/ou d'autres composants agrégats.

Un composant atomique est un composant dont la directive représente une démarche ou une activité qui ne peut plus être décomposée en sous-directives réutilisables, même si c'est une directive tactique ou stratégique.

Un composant ne peut être de type agrégat que si sa directive est complexe (tactique ou stratégique) et ses sous-directives représentées par des composants de méthode. Le composant agrégat est alors composé de plusieurs sous-composants, de la même façon que sa directive a des sous-directives. Ces

sous-composants à leur tour peuvent être soit atomiques soit agrégats. Une méthode est un composant agrégat de plus haut niveau.

Par exemple la directive de la Figure 9 est représentée par un composant atomique. Même si elle est une directive complexe, ses sous-directives ne sont pas définies par des composants de méthodes. En conséquence, le composant comportant cette directive est un composant atomique. La directive de la Figure 10, au contraire, est représentée par un composant agrégat car toutes ses sous-directives sont également représentées par des composants.

3. STRUCTURE D'UN COMPOSANT DE METHODE

Dans cette section on détaille chacun des éléments du méta-modèle introduit à la section 2.

3.1 Notion de directive

Dans cette section nous développons la notion de directive.

Une directive est définie dans [Le Petit Robert 95] comme *"un ensemble des indications sur la façon de procéder pour réaliser un objectif ou exécuter une activité"*. La première option, qui est celle relative à la réalisation d'un objectif, s'applique dans notre cas. Plus précisément, une directive définit la connaissance de la méthode pour guider l'ingénieur d'applications dans la réalisation d'une intention dans une situation donnée. Elle préconise un processus à suivre pour réaliser une intention.

Une directive représente la partie essentielle d'un composant. Elle fournit une démarche de construction du produit cible.

D'une manière générale, toute directive a une *signature* et un *corps* (Figure 7). La *signature* caractérise les conditions dans lesquelles la directive peut être appliquée et le résultat qu'elle permet d'obtenir, sans pour autant dire comment faire. C'est le *corps* définit la démarche à suivre pour satisfaire l'intention capturée dans sa signature.

La *signature* de la directive est définie par un couple <*situation, intention*>. Les conditions d'application de la directive sont capturées dans sa situation. Le résultat auquel elle permet d'aboutir est capturé dans son intention. Par conséquent, chaque directive s'applique dans une situation particulière pour satisfaire une intention particulière. La Figure 14 montre un exemple de directive issue de la méthode OOSE. La signature de cette directive <*(Description du problème), Identifier un cas d'utilisation*> dit que son objectif est d'aider l'ingénieur d'applications à découvrir des cas d'utilisation d'un système (intention) à partir de la description du problème correspondant (situation). Le corps de cette directive contient les recommandations pour atteindre cet objectif.

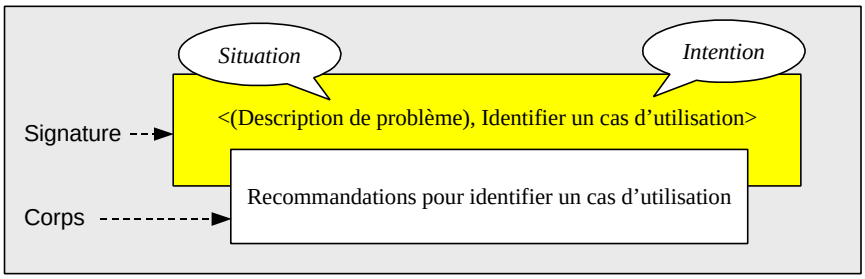


Figure 14 : Exemple de directive issue de la méthode OOSE

Puisque le processus à réaliser pour satisfaire l'intention de la directive est capturé dans le corps de la directive, on peut voir une directive comme une boîte noire qui encapsule le processus de transformation du produit d'entrée en produit de sortie (Figure 15).

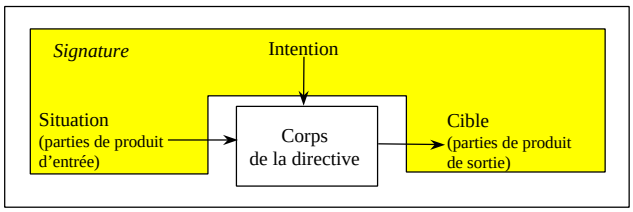


Figure 15 : Vue générale d'une directive

Le produit d'entrée représente la situation dans laquelle la directive peut être appliquée. Il peut être composé de plusieurs parties du produit en construction qui sont nécessaires pour commencer l'exécution du processus encapsulé dans le corps de la directive. L'intention reflète le but à atteindre dans cette situation. Le produit de sortie représente le résultat de l'exécution du processus du composant. C'est la cible de son intention. Selon la structure prédéfinie de but proposée par N. Prat dans [Prat 97], [Prat 99] et qui est développée à la section 3.2.2, l'expression d'un but est composée d'un verbe, d'une cible et d'un certain nombre de paramètres comme le moyen et la manière de satisfaire le but, la source et la destination du but etc. Le fait que la cible soit intégrée dans l'expression de l'intention permet de caractériser le processus de la directive par le couple <situation, intention> appelé la signature de la directive.

La Figure 16 illustre deux exemples de directive. La signature de la première directive *<(But), Ecrire un scénario avec les directives de style>* indique qu'elle s'applique lorsqu'un but a été défini (le produit d'entrée) et que l'intention est de décrire par un scénario (le produit de sortie) le comportement du système permettant de satisfaire ce but. Il est utile de préciser la manière de satisfaire l'intention car en appliquant des stratégies différentes on peut satisfaire la même intention de manière différente. Par exemple, les intentions *"Ecrire un scénario avec les directives de contenu"* et *"Ecrire un scénario avec les directives de style"* représentent le même objectif *"Ecrire un scénario"*, mais les manières de le faire sont différentes. Par conséquent, les processus à exécuter sont différents et ils sont capturés dans des directives différentes. Le deuxième exemple de la Figure 16 représente une directive dont la signature est *<(But), Découvrir un but alternatif au but initial avec une stratégie basée sur le modèle de but>*. La situation de cette directive est la même que précédemment mais l'intention est différente, elle permet de découvrir d'autres buts, ceux qui sont alternatifs au but de la situation. Chacune de ces directives appartient à un composant de méthode différent.

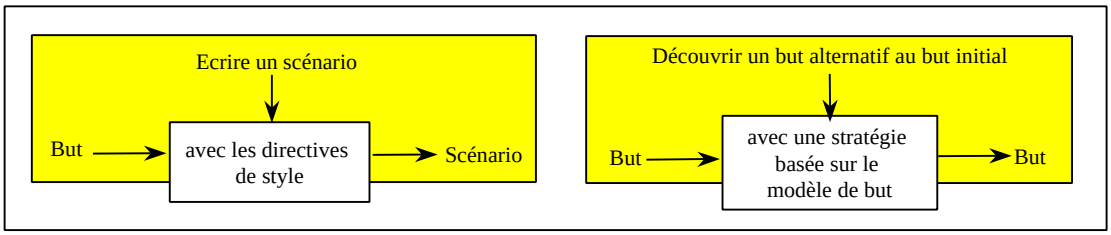


Figure 16 : Exemples des directives

Etant donné que la directive est la partie principale d'un composant, toute la section 4 est consacrée à la définition détaillée de cette notion.

3.2 Signature d'une directive

La signature d'une directive devient la signature d'un composant si la directive est un composant. La signature d'un composant joue un rôle essentiel dans son identification et sa recherche dans la base de composants de méthode car elle définit l'objectif du composant et les conditions de son application.

Une signature est définie par un couple <*situation, intention*> caractérisant le contexte d'application du composant. La Figure 17 illustre la partie du méta-modèle (représenté à la Figure 7) concernant la structure de la signature. Comme le montre cette figure, une situation est basée sur la ou les parties de produit qui sont nécessaires pour satisfaire l'intention du composant.

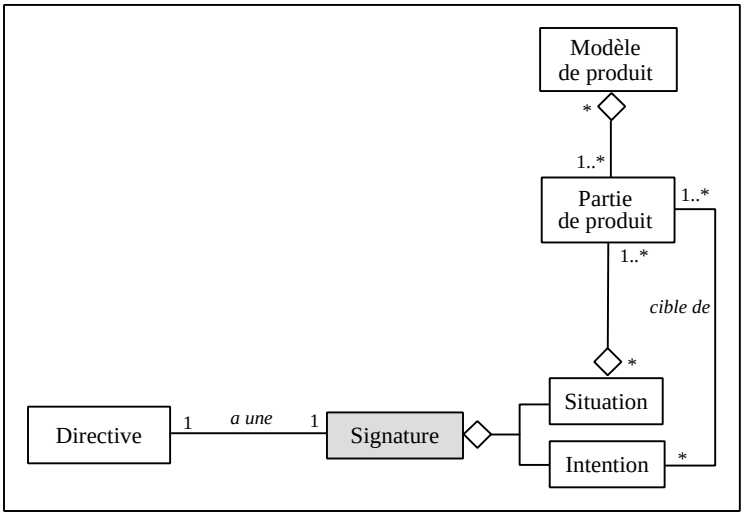


Figure 17 : Structure d'une signature

Une intention exprime un but que l'ingénieur d'applications souhaite atteindre en appliquant le composant. La cible de ce but est composée d'une ou plusieurs parties de produit faisant partie du fragment de produit du composant.

Nous développons plus en détail les concepts de situation et d'intention dans les sous-sections suivantes.

3.2.1 Situation

La situation dans la signature d'une directive identifie une partie de produit en cours de développement nécessaire à la satisfaction de l'intention de la directive (Figure 17). Elle exprime la situation dans laquelle la directive peut être appliquée pour la satisfaction de l'intention qui lui est associée.

Chaque partie de produit référencée dans la situation est un élément du modèle de produit de la méthode. Elle fait partie de l'ensemble des parties de produit associées à la directive. Il peut s'agir d'un élément de produit atomique, d'une association de plusieurs éléments de produit ou même du modèle de produit de la méthode en entier. Nous définissons les concepts de partie de produit et de modèle de produit à la section 3.3 de ce chapitre.

Certaines situations nécessitent de préciser l'état dans lequel doit être chaque partie de produit afin de pouvoir appliquer le composant. L'état d'une partie de produit composant la situation est exprimé avec la *condition d'occurrence*. Par exemple, *<(But avec état(But)=non structuré), Formaliser un but>* est une signature de composant de méthode. La situation de cette signature contient une partie de produit appelée *"But"* qui est précisée par une condition d'occurrence définissant l'état dans lequel doit être le *"But"* pour que l'application du composant soit possible. *"Formaliser un But"* dans cette signature exprime l'intention du composant.

3.2.2 Intention

Une intention est un but, un objectif qu'un ingénieur d'applications à en tête à un moment donné du processus d'ingénierie. Par exemple, *"Construire un modèle des cas d'utilisation"* est une des intentions que l'on peut exprimer dans la démarche de la méthode OOSE.

Pour la représentation d'une intention nous utilisons la structure proposée par N. Prat dans [Prat 97], [Prat 99]. Suivant cette structure, nous représentons une intention par le langage naturel en commençant par un verbe et en ajoutant un ou plusieurs paramètres. Chaque paramètre joue un rôle différent par rapport au verbe. La Figure 18 illustre les différents paramètres d'une intention.

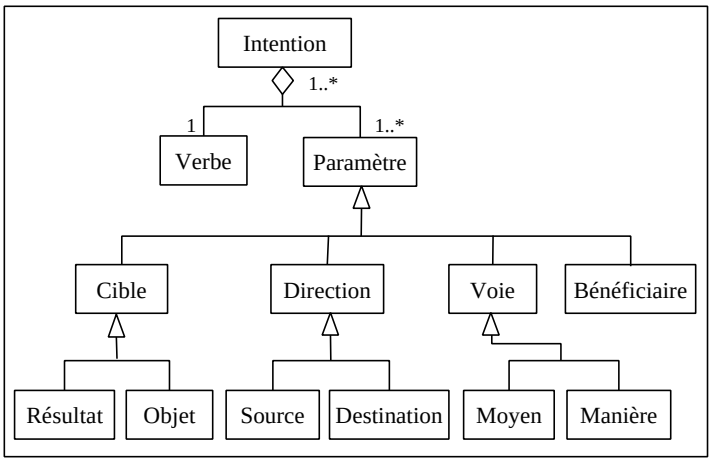


Figure 18 : Structure d'une intention

La *Cible* désigne la ou les entité(s) affectée(s) par l'intention. Dans les exemples suivants *"un scénario"* est la cible des verbes *"Ecrire"* et *"Conceptualise"*.

(a) *Ecrire* $_{verbe}$ *(un scénario)* $_{cible}$ ou plus précisément *Ecrire* $_{verbe}$ *(un scénario)* $_{résultat}$

(b) *Conceptualiser* $_{verbe}$ *(un scénario)* $_{cible}$ ou plus précisément *Conceptualiser* $_{verbe}$ *(un scénario)* $_{objet}$

Comme le montrent les exemples (a) et (b) ci-dessus, il y a deux types de cible, l'*objet* et le *résultat.* Les deux éléments référencent les parties de produit qui sont soit des objets soit des résultats de l'intention. Un objet est supposé exister avant la réalisation de l'intention. Par exemple dans (b), *"un scénario"* est un objet de l'intention *"Conceptualiser"*. Par contre, un résultat découle de la réalisation de l'intention. Ainsi, dans (a), la cible *"un scénario"* est le résultat de la réalisation de l'intention *"Ecrire"*. La cible peut dans certains cas être une intention. Par exemple, dans *"Progresser* $_{verbe}$ *(vers Découvrir un but)* $_{cible}$*"* la cible du verbe *"Progresser"* est une intention *"Découvrir* $_{verbe}$ *(un but)* $_{cible}$*"* qui est aussi décrite par un verbe et une cible.

Deux types de *direction* appelés la *source* et la *destination* identifient respectivement l'endroit initial et final de l'objet.

La *Source* identifie l'emplacement initial des objets qui sont utilisés.

(c) *Découvrir* $_{verbe}$ *(un but)* $_{résultat}$ *(à partir d'un scénario)* $_{source}$

(d) *Progresser* $_{verbe}$ *(de Ecrire un scénario)* $_{source}$

Dans le cas (c), la source de l'intention *"Découvrir"* est *"un scénario"*. La source peut, dans certains cas, être une intention. Ainsi, dans l'expression (d) ci-dessus la source du verbe *"Progresser"* est une intention *"Ecrire un scénario"*.

La *Destination* identifie l'emplacement des objets après leur utilisation. Dans l'intention (e) *"le dictionnaire"* est la destination de la cible *"un objet"*.

(e) *Décrire* $_{verbe}$ *(un objet)* $_{cible}$ *(dans le dictionnaire)* $_{destination}$

Deux types de voie appelés le *moyen* et la *manière* précisent comment l'intention est réalisée.

Le *Moyen* décrit l'entité qui sert d'instrument pour atteindre l'intention.

(f) *Conceptualiser* $_{verbe}$ *(un scénario)* $_{objet}$ *(avec un outil linguistique)* $_{moyen}$

Par exemple, dans l'intention (f), *"un outil linguistique"* est un moyen pour satisfaire l'intention *"conceptualiser un scénario"*.

La Manière identifie comment l'intention peut être satisfaite.

(g) *Ecrire* $_{verbe}$ *(un scénario)* $_{résultat}$ *(avec les directives de style)* $_{manière}$

(h) *Décrire* $_{verbe}$ *(les besoins fonctionnels du système)* $_{objet}$ *[en écrivant* $_{verbe}$ *(un scénario)* $_{objet}$ *(avec les directives de style)* $_{manière}$ *]* $_{manière}$

Dans l'intention (g), *"avec les directives de style"* est la manière pour satisfaire le but *"Ecrire un scénario"*. Une manière, quand elle est complexe, peut être exprimée comme une intention. Par conséquent, une intention peut être définie récursivement. L'intention du cas (h) comprend une définition récursive de la manière *"[en écrivant $_{verbe}$ (un scénario) $_{objet}$ (avec les directives de style) $_{manière}$] $_{manière}$"* qui est elle-même une intention comprenant le verbe *"Ecrire"* et les paramètres cible et manière.

Finalement, le *Bénéficiaire* est une personne (ou un groupe de personnes) pour qui l'intention doit être satisfaite. Dans l'exemple (i) ci-dessous, *"l'ingénieur d'applications"* est un bénéficiaire de l'intention *"Afficher les directives de style"*.

(i) *Afficher $_{verbe}$ (les directives de style) $_{résultat}$ (à l'ingénieur d'applications) $_{bénéficiaire.}$*

3.3 Partie de produit d'un composant de méthode

Comme nous l'avons déjà mentionné dans la définition d'un composant de méthode, la partie produit d'un composant définit le produit qui va être obtenu ou utilisé en exécutant sa partie processus, c'est-à-dire la directive. Le composant intègre plusieurs parties de produit qui sont des éléments du modèle de produit de sa méthode.

Puisque le modèle de produit d'une méthode est une instance de méta-modèle de produit, nous présentons d'abord ce méta-modèle de manière détaillée et nous proposons ensuite un exemple d'instanciation.

3.3.1 Méta-modèle de produit

La Figure 19 représente le méta-modèle de produit inspiré de [Plihon 96].

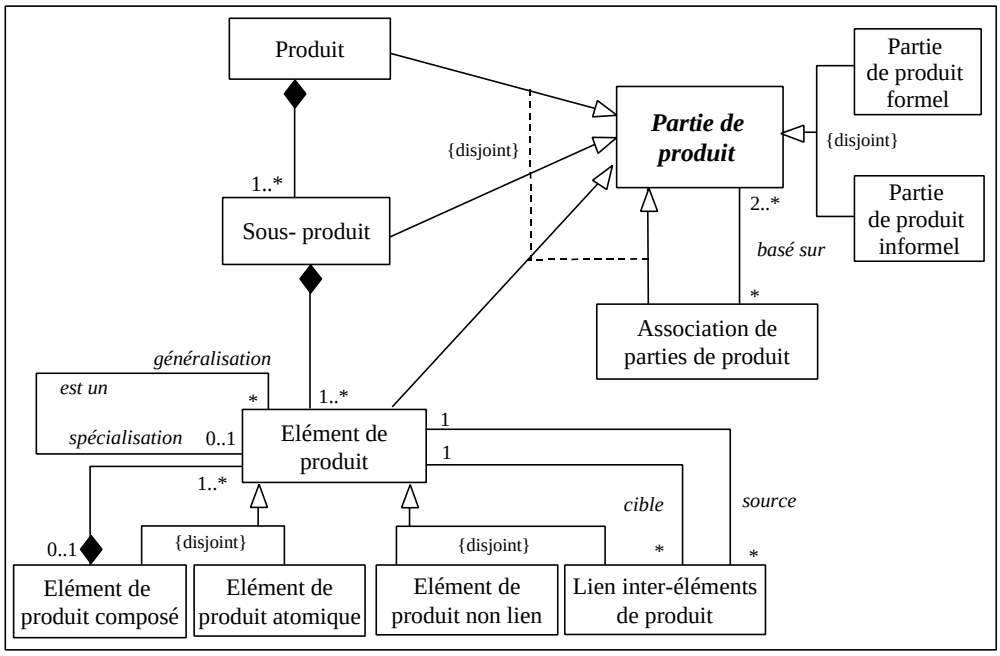


Figure 19 : Méta-modèle de produit

Le concept central de ce méta-modèle est celui de *partie de produit.* Une partie de produit représente n'importe quel fragment de connaissance se rapportant au produit. Il peut s'agir de la totalité du produit en cours de construction, représenté par le concept de *produit*, ou d'un fragment de granularité plus fine représenté par un sous-produit, un *élément de produit* ou une *association de parties de produit.* Ainsi, comme le montre la Figure 19, le concept de *partie de produit* est spécialisé en *produit, sous-produit, élément de produit,* et *association de parties de produit*.

Le *produit* est le résultat attendu du processus. Par exemple, un schéma conceptuel est le résultat d'un processus d'analyse. Un produit peut être composé de *sous-produits*. Dans la méthode OOSE, le résultat du processus d'analyse est composé de deux sous-produits qui sont le *"modèle des cas d'utilisation"* et le *"modèle d'analyse"*. Un sous produit peut être décomposé à son tour en éléments de granularité plus fine appelés *éléments de produit.* Le sous-produit *"modèle des cas d'utilisation"* par exemple, peut être décomposé avec les concepts *"acteur"* et *"cas d'utilisation"* représentant des éléments de produit. Les *associations de parties de produit* correspondent à des combinaisons arbitraires de *parties de produit* appartenant à des *produits* distincts.

Deux caractéristiques, le type et la complexité de l'élément, permettent de spécialiser la notion d'élément de produit en deux partitions. La partition selon le type de l'élément de produit permet de faire la distinction entre les *liens inter-éléments de produit* et les *éléments de produit non lien* tandis que la partition selon la complexité de l'élément de produit permet de distinguer les *éléments de produit atomiques* des *éléments de produit composés.*

Les *éléments de produit non lien* représentent des concepts du produit. Ils ont une existence propre. Le *"scénario"* et l'*"acteur"*, par exemple, sont considérés comme des éléments de produit non lien dans la méthode OOSE. Chacun de ces éléments de produit a un sens indépendamment des autres.

Les *liens inter-éléments de produit* traduisent une relation existant entre deux éléments de produit. On les distingue des éléments de produit non lien car, si on les considère seul, sans leurs extrémités, ils n'ont pas de sens. Le *lien de composition* liant un *"cas d'utilisation"* et un *"scénario"* dans le modèle de produit de OOSE est un exemple typique de *lien inter-éléments de produit.* Ce lien considéré seul n'est pas significatif.

Les *éléments de produit atomiques* ne sont pas décomposables. Ils sont créés, modifiés et supprimés par des actions atomiques. Un attribut, une opération etc. peuvent être considérés comme des éléments de produit atomiques.

Les *éléments de produit composés* se décomposent en éléments de produit, pouvant être eux-mêmes atomiques ou composés. Le modèle de produit du *"modèle des cas d'utilisation"* de la méthode OOSE, par exemple, est considéré comme un élément de produit composé puisqu'il peut être décomposé en éléments de produit plus fins qui sont des *"cas d'utilisation"* et des *"acteurs"*. Les *"cas d'utilisation"* sont à leur tour composés d'autres éléments qui sont des *"scénarios"*. Ceci montre que la décomposition d'éléments *de produit composé* peut être appliquée récursivement et porter sur plusieurs niveaux.

Les *éléments de produit* peuvent participer à des hiérarchies d'héritage, comme l'exprime la relation *est-un* sur la classe *élément de produit*, à la Figure 19. Dans OOSE, les concepts *"scénario normal"* et *"scénario exceptionnel"* héritent du concept *"scénario"*.

Enfin, une *partie de produit* est caractérisée comme formelle ou non formelle. Une partie de produit formelle est décrite en utilisant les concepts d'un modèle, comme par exemple le *"but"* dans l'approche CREWS-*L'Ecritoire*, contrairement à une partie de produit informelle qui est décrite en langage naturel comme, par exemple, le concept de *"scénario"* dans OOSE.

3.3.2 Exemples de parties de produit d'un composant

Prenons comme exemple les directives représentée à la Figure 16. Les parties de produit associées à la première directive, dont la signature est *<(But), Ecrire un scénario avec les directives de style>*, définissent la structure des scénarios que l'on obtient en appliquant la directive. Elles sont illustrées à la Figure 20 (a). La partie de produit associée à la deuxième directive, dont la signature est *<(But), Découvrir un but alternatif au but initial avec la stratégie de découverte basée sur la structure de but>* définit la structure de but. Elle est illustrée à la Figure 20 (b).

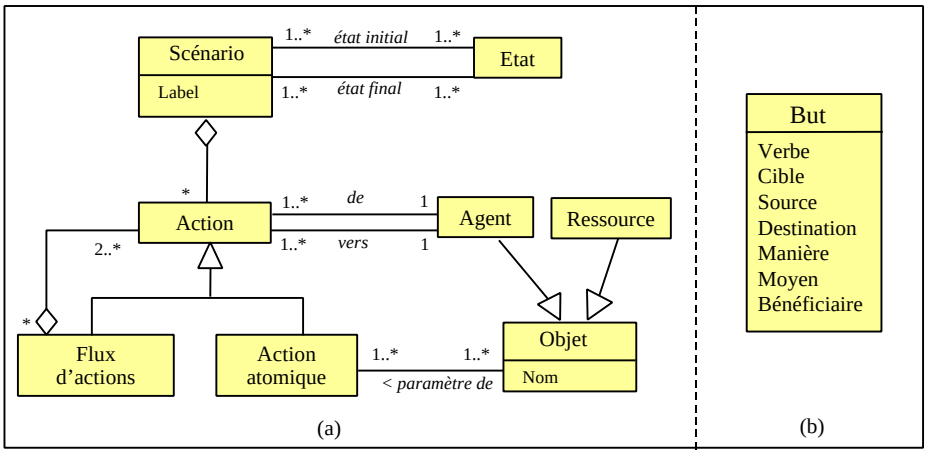


Figure 20 : Exemples de parties de produit

Puisque les deux directives sont des éléments de la démarche de la méthode CREWS-*L'Ecritoire*, les parties de produit utilisées par ces directives sont des éléments de modèle de produit de cette même méthode. La Figure 21 illustre comment les parties de produit utilisées par la première directive instancient le méta-modèle de produit défini à la section 3.3.1.

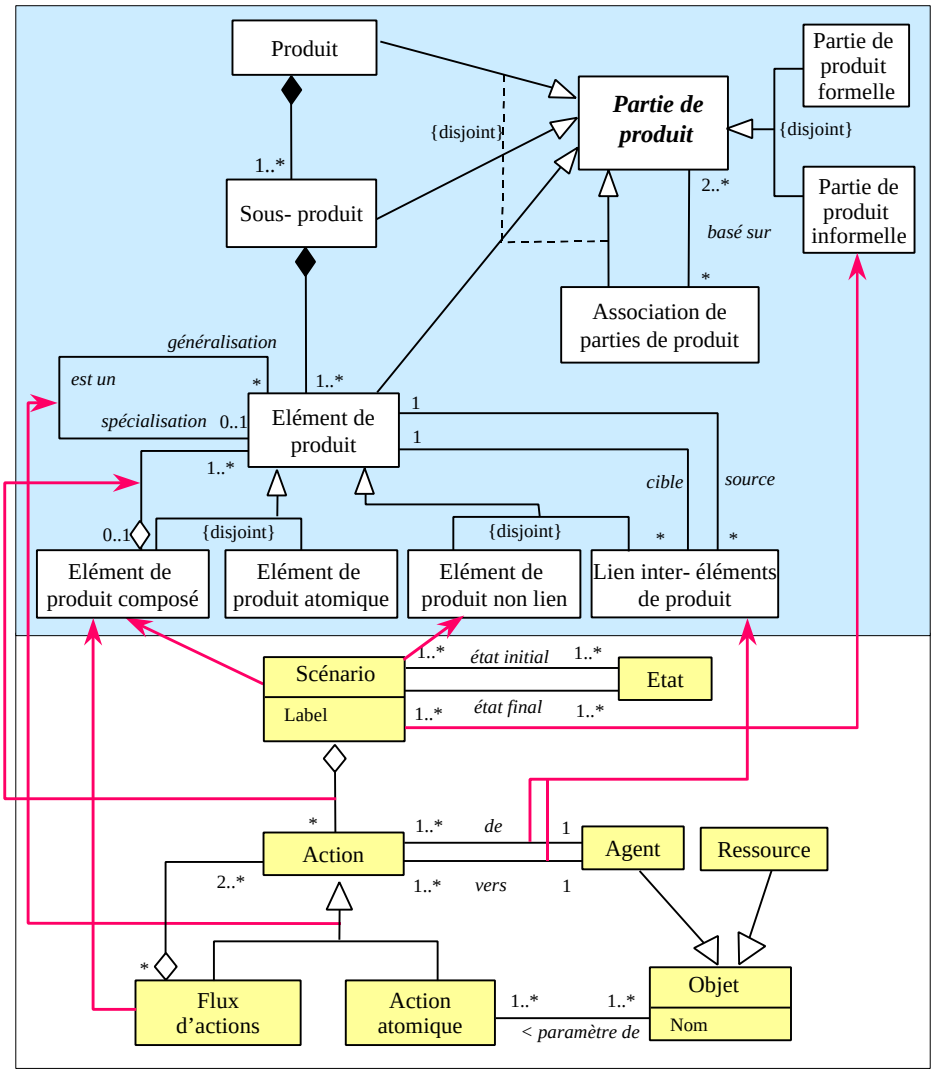


Figure 21 : Exemple d'instanciation du méta-modèle de produit

Comme le montre la Figure 21, le concept *"scénario"* est une instance du concept *élément de produit*. Il est considéré comme un *élément de produit composé* puisqu'il est composé d'*éléments de produit* plus fins qui sont des *"actions"*. Cette relation de composition est instance de la relation *composé de* entre *élément de produit composé* et *élément de produit*. Plus précisément, le *"scénario"* est un *élément de produit composé* selon sa structure, car il est composé à son tour d'autres éléments de produit qui sont des *"actions"*. Il est aussi à la fois un *élément de produit non lien* selon son type et une *partie de produit informelle*. Les relations *"de"* et *"vers"* entre le concept *"action"* et le concept *"agent"* sont instances du concept *lien inter-éléments de produit*. Le concept *"agent"* est un *élément de produit atomique*.

Selon la taille et la richesse de la directive, l'ensemble des parties de produit nécessaires à son exécution peut représenter tout le modèle de produit de sa méthode d'origine, ou seulement certaines parties de ce modèle. Puisque le même modèle de produit est partagé par plusieurs composants issus de la même méthode, certaines parties de produit peuvent être retrouvées dans plusieurs composants de méthode. Par exemple, la méthode CREWS-*L'Ecritoire* propose deux composants supportant l'écriture des scénarios, chacun proposant une stratégie différente. L'un d'entre eux permet d'écrire

des scénarios avec *"les directives de style"* et l'autre avec *"les directives de contenu"*. Puisque la structure des scénarios à écrire dans les deux cas reste la même, les deux composants partagent les mêmes parties de produit - celles décrivant la structure d'un scénario.

3.4 Descripteur d'un composant

3.4.1 Vue générale sur un descripteur

Etant donné qu'un composant de méthode est une brique de construction réutilisable dans la définition de nouvelles méthodes, on doit pouvoir le stocker dans une base de composants et pouvoir l'extraire de cette base quand c'est nécessaire. La question que l'on se pose devant une base de composants de méthode est *"comment extraire le plus facilement possible les composants correspondant aux besoins de l'ingénieur de méthodes dont l'objectif est de construire une méthode ou d'enrichir une méthode existante."* Pour répondre à cette question nous avons besoin de définir où, pourquoi et comment le composant spécifique peut être réutilisé, c'est-à-dire définir le contexte d'utilisation du composant.

En matière de représentation de connaissance réutilisable, il est essentiel de bien distinguer la connaissance qui supporte la réutilisation et la connaissance effectivement réutilisée. De la richesse de la première dépend la réutilisabilité de la seconde. La connaissance que l'on réutilise effectivement c'est le composant lui-même, c'est-à-dire une directive avec les parties de produit associées. La connaissance pour réutiliser, c'est-à-dire la connaissance sur le contexte de réutilisation du composant est capturée dans son *descripteur* [Rolland 98c], [Ralyté 99b].

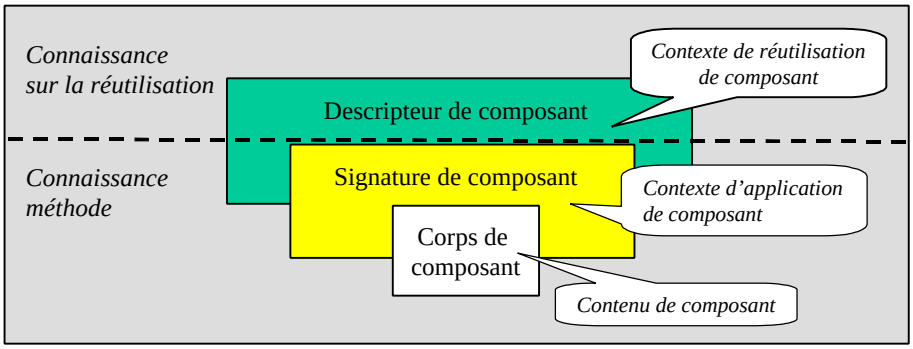


Figure 22 : Vue générale d'un composant de méthode

La Figure 22 illustre la vue générale d'un composant de méthode. Selon cette illustration, la connaissance méthode est capturée dans le corps du composant et sa signature, tandis que la connaissance sur le contexte de réutilisation est définie dans son descripteur. En conséquence, le concept de *descripteur* permet de formaliser et de stocker l'information sur le contexte de l'utilisation d'un composant. Dans le processus de réutilisation des composants, les descripteurs sont utilisés dans l'étape de leur extraction de la base alors que la connaissance méthode capturée dans le composant lui-même est réutilisée effectivement et intégrée dans une nouvelle méthode.

Nous utilisons la notion de descripteur [De Antonnellis 91] comme un moyen pour décrire les composants. Notre concept de descripteur est similaire à l'un des schémas de classification à base de facettes [Prieto-Diaz 87] développé dans le contexte de la réutilisation de logiciels.

3.4.2 Structure d'un descripteur

Le processus d'extraction des composants de la base est contextuel. En effet, l'utilisateur est confronté à une certaine situation et désire réaliser une certaine intention. Par conséquent, le descripteur cherche à capturer la situation dans laquelle le composant pourrait être réutilisé et l'intention qu'il aiderait à satisfaire. Il étend la vue contextuelle utilisée dans la signature du composant pour représenter la connaissance sur la réutilisation de celui-ci.

La Figure 23 représente la structure d'un descripteur dont les deux facettes principales sont la *situation de réutilisation* et l'*intention de réutilisation*. Chaque composant de méthode vise à soutenir une activité spécifique de la conception des systèmes par un chemin particulier. La situation du descripteur fait référence à cette activité de conception tandis que l'intention exprime le but de la conception associé à cette activité. Le descripteur est contextuel car il comporte la connaissance situationnelle et intentionnelle sur le contexte de la réutilisation d'un composant de méthode. Nous développons les concepts de la *situation de réutilisation* et de l'*intention de réutilisation* dans les sous-sections qui suivent.

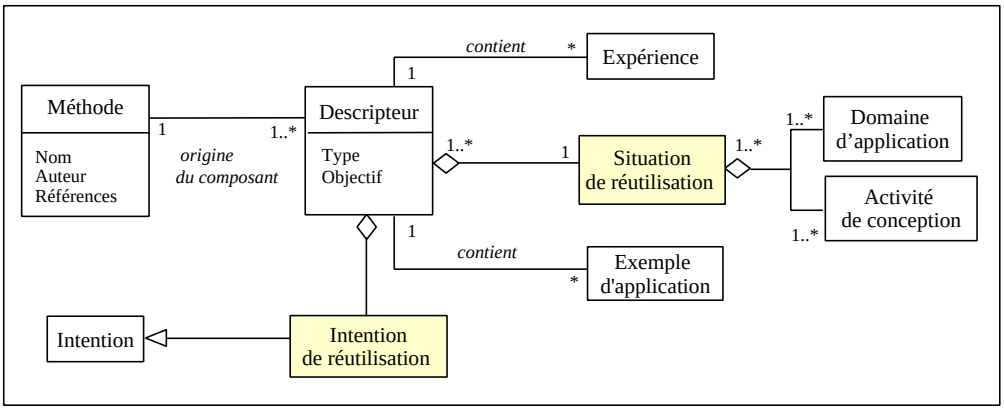


Figure 23 : Structure d'un descripteur

Le descripteur décrit en quelques mots *l'objectif* du composant, ce qui permet de sélectionner ou de rejeter le composant sans regarder son contenu. Il détermine aussi le *type* du composant : atomique ou agrégat, ce qui permet d'affiner la recherche d'un composant dans la base. Si celui-ci est de type agrégat, nous pouvons affiner la recherche en analysant sa structure et en sélectionnant l'un de ses sous-composants. Si, au contraire, il est de type atomique, il ne pourra pas être décomposé, mais par contre, nous pouvons chercher les agrégat possibles incluant ce composant en tant que sous-composant.

L'origine du composant est aussi capturée dans son descripteur. Elle contient l'information sur la méthode ou l'approche qui est à la base du composant. Le nom de la méthode, les auteurs, les références dans la littérature peuvent être utiles dans l'identification de celui-ci. Par exemple, grâce à

cette facette nous pouvons retrouver la méthode d'origine du composant. De la même manière, lorsque l'on a une méthode, nous pouvons retrouver tous ses composants (tous les composants définis à partir de la méthode CREWS-*L'Ecritoire* par exemple).

La facette *Expérience* contient l'information sur l'utilisation du composant dans des applications concrètes. Après chaque utilisation d'un composant, l'ingénieur de développement peut décrire son expérience et donner des astuces d'application ou des schémas d'assemblage avec d'autres composants.

La facette *Exemple d'application* comporte des cas d'application du composant.

3.4.2.1 Situation de réutilisation

La *situation de réutilisation* inclut deux aspects : le *domaine d'application* dans lequel le composant peut être appliqué et *l'activité de conception* dans laquelle le composant est pertinent. Certains composants peuvent être appliqués dans plusieurs domaines et peuvent servir dans plusieurs activités de conception.

Par exemple, le composant de la méthode CREWS-*L'Ecritoire*, dont la signature est *<(But), Ecrire un scénario en prose libre>* aide l'écriture des scénarios en proposant des directives de style et/ou de contenu. Chacun des scénarios en question doit décrire le comportement du système face à un but donné. Nous pouvons nous servir de ce composant dans différents domaines d'application comme les *systèmes d'information*, les *interfaces homme-machine*, l'*organisation d'entreprise*, les *systèmes socio-techniques*, etc. dans des activités de développement comme la *découverte des besoins*, les *tests du système*, la *documentation d'utilisation du système*. Par conséquent, la situation du descripteur de ce composant serait : *Domaine d'application : systèmes d'information, interface homme-machine, organisation d'entreprise, systèmes socio-techniques ; Activité de développement : découverte des besoins, test du système, documentation d'utilisation du système*.

3.4.2.2 Intention de réutilisation

L'intention de réutilisation exprime comment le processus encapsulé dans le composant participe à l'exécution de l'activité de développement de la situation.

L'intention de réutilisation a la même structure que l'intention de la signature (voir la section 3.2.2), c'est-à-dire qu'elle est composée d'un verbe, d'une cible et de plusieurs paramètres, comme la manière, le moyen, le bénéficiaire, etc. La particularité de l'intention du descripteur est la suivante : le paramètre manière définit d'une manière récursive l'intention de la signature du composant. Prenons comme exemple le composant utilisé dans la section précédente. L'intention de réutilisation de ce composant est exprimée comme suit : *"Documenter les besoins du système en écrivant les scénarios en prose libre"*. Cette intention est composée du verbe *"Documenter"*, de la cible *"les besoins du système"* et de la manière *"en écrivant les scénarios en prose libre"* qui est elle-même une manière complexe. Elle exprime l'intention du composant correspondant et est composée également du verbe *"Ecrire"*, de la cible *"un scénario"* et de la manière simple *"en prose libre"*. Par conséquent,

l'intention de réutilisation est exprimée de la manière suivante : *"Documenter$_{verbe}$ (les besoins du système)$_{cible}$ (en écrivant $_{verbe}$ (les scénarios)$_{cible}$ (en prose libre)$_{manière}$)$_{manière}$"*.

3.4.3 Exemples de descripteurs

Nous proposons maintenant quelques exemples de descripteurs. La Figure 24 illustre le descripteur d'un composant issu de la méthode CREWS-*L'Ecritoire* dont la signature est *<(But), Ecrire un scénario en prose libre>*.

***Objectif** : L'objectif de ce composant est d'aider l'ingénieur des besoins dans l'écriture de scénarios. Le composant aide à écrire un scénario décrivant comment un but du système défini préalablement peut être satisfait par le système.*
***Type** : Agrégat*
***Situation de réutilisation** :*
***Domaine d'application** : Systèmes d'information, Interface Homme-Machine, Réorganisation des processus d'entreprise, Système socio-technique*
***Activité de conception** : Découverte des besoins, Test du système, Documentation d'utilisation du système*
***Intention de réutilisation**: Documenter les besoins du système en écrivant des scénarios en prose libre.*
***Origine** :* CREWS-*L'Ecritoire*

Figure 24 : Exemple de descripteur d'un composant issu de la méthode CREWS-*L'Ecritoire*

Comme nous le montre cette illustration, le composant sert à documenter les besoins du système par des scénarios. Il peut être appliqué dans plusieurs domaines d'applications pour les activités de découverte des besoins, de test et de documentation du système. Le descripteur précise également que c'est un composant agrégat. De plus, nous savons que le composant est issu de la méthode CREWS-*L'Ecritoire*.

La Figure 25 illustre un autre exemple de descripteur.

***Objectif** : L'objectif de ce composant est de guider l'ingénieur des besoins dans la découverte des solutions de conception alternatives. Un mécanisme de génération des nouveaux buts/besoins est fourni avec le composant. On appelle ces nouveaux buts des buts alternatifs car ils capturent les solutions alternatives à la solution de conception initiale.*
***Type** : Agrégat*
***Situation de réutilisation**:*
***Domaine d'application** : Systèmes d'information, Interface Homme-Machine, Réorganisation des processus d'entreprise*
***Activité de conception** : Découverte des besoins*
***Intention de réutilisation** : Découvrir les besoins de système en élucidant des buts alternatifs par application du modèle de but*

Origine *:* CREWS-*L'Ecritoire*

Figure 25 : Exemple de descripteur d'un composant issu de la méthode CREWS-*L'Ecritoire*

Selon ce descripteur, le composant aide à découvrir des besoins de système sous forme de buts qui représentent des solutions alternatives de la conception. Le composant peut être utilisé dans plusieurs domaines d'application.

4. DEFINITION DETAILLEE DE LA NOTION DE DIRECTIVE

Cette section est consacrée à la description détaillée de la notion de directive.

Suivant le raisonnement décrit à la section précédente, la directive d'un composant a deux parties : une signature et un corps. La signature représente la partie visible de la directive. Quand au corps, il explique comment procéder pour satisfaire l'intention dans une situation donnée. Le corps fournit des conseils qui guident l'exécution du processus et relient le processus avec les parties de produit impliquées.

Rappelons-nous que la signature d'une directive est définie par un couple <situation, intention> caractérisant le contexte d'application de la directive et donc le contexte d'application du composant. La situation d'une directive est toute partie de produit en cours de développement nécessaire à la satisfaction de l'intention de cette directive. La signature exprime la situation dans laquelle la directive peut offrir du guidage pour la satisfaction de l'intention qui lui est associée. L'intention de la directive est un but, un objectif qu'un ingénieur d'applications désire réaliser à un moment donné du processus d'ingénierie. Par exemple, *"Construire un modèle des cas d'utilisation"* est une des intentions que l'on peut exprimer dans la démarche de la méthode OOSE.

Le corps de la directive exprime explicitement le guidage fourni par celle-ci. Il contient un ensemble d'instructions définissant comment procéder pour satisfaire l'intention définie dans la signature de la directive. Il propose une ou plusieurs démarches à suivre pour aboutir au résultat attendu. Le corps de la directive représente la partie interne alors que la signature représente l'interface de la directive.

Plusieurs types de représentation peuvent être utilisés pour capturer le corps de la directive. Comme on l'a montré à la Figure 7, nous proposons trois types de directive en fonction de leur complexité, de leur richesse et de la manière dont elles sont exprimées. Nous les appelons *directive simple*, *directive tactique* et *directive stratégique*.

- Une *directive simple* est une directive qui n'est pas structurée. Elle est soit informelle soit exécutable. Une directive informelle explique de manière textuelle comment procéder pour obtenir le produit cible tandis qu'une directive exécutable propose une action à exécuter, soit avec un outil soit de manière manuelle.

- Une *directive tactique* a une structure d'arbre. C'est une directive complexe composée d'autres directives, soit sous forme d'un plan soit sous forme d'un choix de plusieurs sous-directives

alternatives. Le formalisme utilisé dans ce type de directive est inspiré du formalisme de modélisation des processus NATURE [Rolland 95], [Grosz 97], [Jarke 99]. Ce dernier est basé sur la notion du *contexte* et représente le processus de développement par une hiérarchie de contextes.

- Une *directive stratégique* a une structure de graphe. Elle représente une vue stratégique de la démarche de développement du produit basée sur un ensemble d'intentions à satisfaire et un ensemble de stratégies pour satisfaire ces intentions. La directive stratégique permet d'exprimer le processus de développement en proposant plusieurs chemins possibles pour satisfaire son intention. Elle est représentée sous forme d'une carte et d'un ensemble de directives associées [Rolland 99], [Benjamen 99].

Nous développons chaque type de directive dans les sections suivantes.

4.1 Directive simple

Elle ne peut pas être décomposée en sous-directives. Comme nous l'avons déjà dit dans la section 2.2.2, une directive simple peut être représentée comme un composant de méthode si elle est réutilisable en tant que brique de construction dans la définition d'une nouvelle méthode. Elle peut aussi être tout simplement un élément d'une autre directive, plus importante. Comme elle n'a pas de structure, une directive simple est un élément atomique dans la démarche d'une méthode.

La Figure 26 définit une directive simple qui peut être informelle ou exécutable.

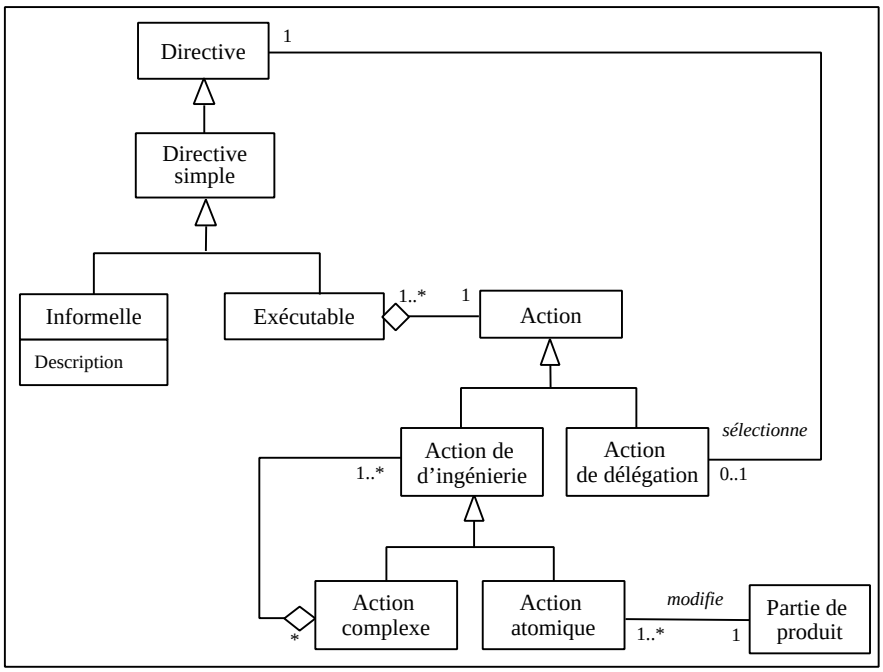


Figure 26 : Structure d'une directive simple

Comme le montre la figure ci-dessus, une directive informelle est associée à une description textuelle tandis qu'une directive exécutable est représentée par une action qui peut être soit atomique soit complexe. Nous développons ces deux types de directives dans les sous-sections suivantes.

4.1.1 Directive informelle

La plupart des méthodes et des approches proposées dans la littérature ne sont pas définies d'une manière formelle. Elles utilisent rarement la méta-modélisation pour définir le produit et la démarche de la méthode. Même si leurs modèles de produit sont présentés avec un niveau de détail suffisant pour être formalisé, les modèles de processus sont souvent omis ou décrits d'une manière informelle sous forme d'hypothèses et de conseils à suivre. Nous proposons d'utiliser la directive informelle pour représenter la démarche d'un composant de méthode qui ne peut pas être formalisée ou exécutée par un outil.

Une directive informelle ne propose pas de démarche détaillée à suivre pour aboutir au résultat attendu. Elle définit seulement des hypothèses, des règles, des conditions à respecter et des contraintes à ne pas violer. Elle peut aussi définir quel est le type du résultat à obtenir sans préciser pour autant comment procéder formellement pour l'obtenir. Des conseils et des exemples peuvent être proposés à l'ingénieur d'applications pour le guider dans la satisfaction de l'intention.

La Figure 27 illustre un exemple de directive issue de la méthode CREWS-*L'Ecritoire*. Sa signature indique qu'elle s'applique dans la situation où l'ingénieur d'applications a identifié un but de manière informelle et que son objectif est de formaliser l'expression de ce but. Le corps de cette directive est présenté de manière informelle.

Signature : *<(But avec état(But)=informel), Formaliser un But suivant la structure prédéfinie>*

Corps : *L'expression du but formalisé doit avoir la structure suivante: il doit être exprimé par un verbe suivi par au moins un paramètre représentant la cible du verbe. Par exemple, le but "Fournir de l'argent aux clients de la banque en utilisant un distributeur automatique de billets" peut être formalisé de manière suivante :*

Fournir $_{verbe}$ *(de l'argent)* $_{cible}$

Si c'est possible, d'autres paramètres comme le moyen ou la manière de satisfaire l'intention, la source et la destination de l'intention doivent être ajoutés pour préciser le but. Par exemple, le même but peut être exprimé de la manière suivante :

Fournir $_{verbe}$ *(de l'argent)* $_{cible}$ *(au client de la banque)* $_{bénéficiaire}$ *(à partir d'un compte dépôt)* $_{source}$ *(avec un distributeur automatique de billets à carte)* $_{moyen}$

Figure 27 : Exemple de directive informelle

4.1.2 Directive exécutable

Une directive exécutable (Figure 26) correspond à une intention qui peut être concrétisée par une action de transformation du produit ou une action de sélection d'une autre directive. Ces deux types d'action sont modélisés comme des types spécialisés du concept action, *action d'ingénierie* et *action de délégation*. L'action *d'ingénierie* consiste à modifier le produit en cours de développement tandis que l'action de *délégation* consiste à déléguer la réalisation d'une intention à une autre directive. Les actions sont exécutées par un outil, une procédure ou un agent humain.

Exécuter une action d'ingénierie modifie une ou plusieurs parties du produit associées à la directive et peut générer une nouvelle situation qui est elle-même sujette à de nouvelles intentions. Par exemple, une action *"Découvrir un but"* conduit à introduire un nouveau but dans le modèle but/scénario de la méthode CREWS-*L'Ecritoire* qui servira ensuite de situation à une autre directive, par exemple *<(But), Ecrire un scénario>*.

L'action de délégation consiste à sélectionner une autre directive. La directive sélectionnée peut être une directive simple, une directive tactique ou une directive stratégique. Ce type d'action est utilisé dans les directives de progression associées à la carte de processus que nous présentons à la section 4.3.2. Leur rôle est de sélectionner une directive appropriée à la réalisation d'une intention. Par exemple, pour réaliser l'intention *"Sélectionner (<(But), Ecrire un scénario avec des directives de style>)"*, il faut exécuter l'action de sélection *"Sélectionner la directive <(But), Ecrire un scénario avec des directives de style>"*.

Une directive exécutable est rarement représentée par un composant de méthode. Elle fait partie d'une ou plusieurs directives tactiques ou stratégiques qui, elles peuvent être des composants.

4.2 Directive tactique

Certaines méthodes proposent des démarches de développement de leurs produits d'une manière assez détaillée pour que l'on puisse les formaliser en utilisant le méta-modèle proposé. Nous proposons de représenter le processus d'une telle méthode sous forme d'une directive tactique.

La structure de directive tactique proposée dans ce mémoire est inspirée de l'approche contextuelle de la modélisation des processus NATURE [Rolland 95], [Jarke 99]. Cette approche représente le modèle de processus d'une méthode au moyen de la notion de *contexte* et d'*arbre de contextes* [Plihon 96], [Rolland 94c], [Plihon 94], [Plihon 95]. La typologie de contextes proposée dans cette approche est appliquée dans notre méta-modèle et nous permet de spécialiser les directives tactiques en directives de deux types : *plan* et *choix*.

La Figure 28 montre notre typologie de directives tactiques en faisant apparaître que celles-ci sont structurées en une hiérarchie de directives de n'importe quel type. Les deux types de directive tactique sont détaillés dans les sous-sections suivantes.

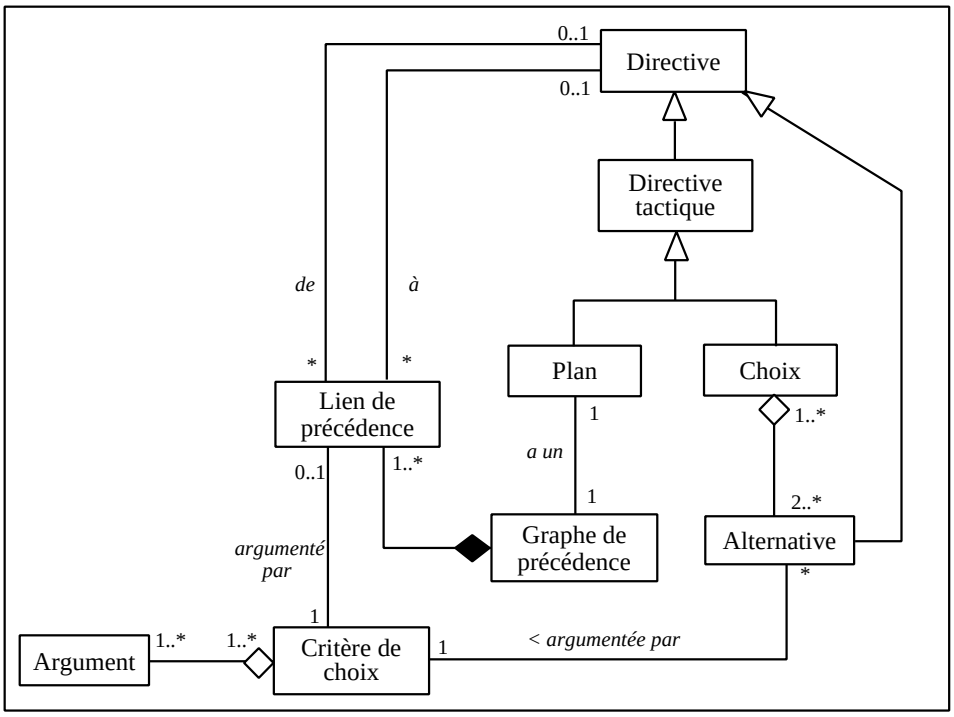


Figure 28 : Structure d'une directive tactique

4.2.1 Directive choix

Une directive choix correspond à une situation qui nécessite l'exploration de différentes possibilités alternatives (Figure 28). Il y a des situations dans lesquelles l'ingénieur d'applications a différentes façons d'atteindre le but qu'il poursuit. Il doit faire un choix parmi un ensemble de possibilités permettant de résoudre le problème. Chaque solution alternative est décrite par une nouvelle directive qui permet de satisfaire la même intention que celle de la directive choix. Ces directives alternatives peuvent appartenir à l'un des trois types possibles : simple, tactique ou stratégique.

La directive choix permet de décomposer une intention en sous-intentions alternatives et d'affiner ainsi l'intention de haut niveau en intentions plus fines. Les alternatives d'une directive choix précisent l'intention de cette dernière, soit en ajoutant une information sur l'approche suivie pour mettre en œuvre l'intention, soit en décrivant les différentes transformations du produit qui peuvent être menées pour atteindre l'objectif de la directive choix. Le lien hiérarchique entre les directives est appelé le *lien de choix* à la Figure 7.

L'ingénieur d'applications explore les différentes possibilités pour la résolution d'un problème à l'aide des *critères de choix*. Ils aident l'ingénieur d'applications à choisir l'alternative la mieux appropriée aux caractéristiques de la situation qu'il traite. Un *critère de choix* est une combinaison d'*arguments* en faveur ou en défaveur du choix d'une alternative. Les arguments sont basés sur des heuristiques ou sur les caractéristiques du produit en cours de développement. Ils sont atomiques et peuvent être réutilisés dans plusieurs critères de choix. Ils sont décrits en langage naturel.

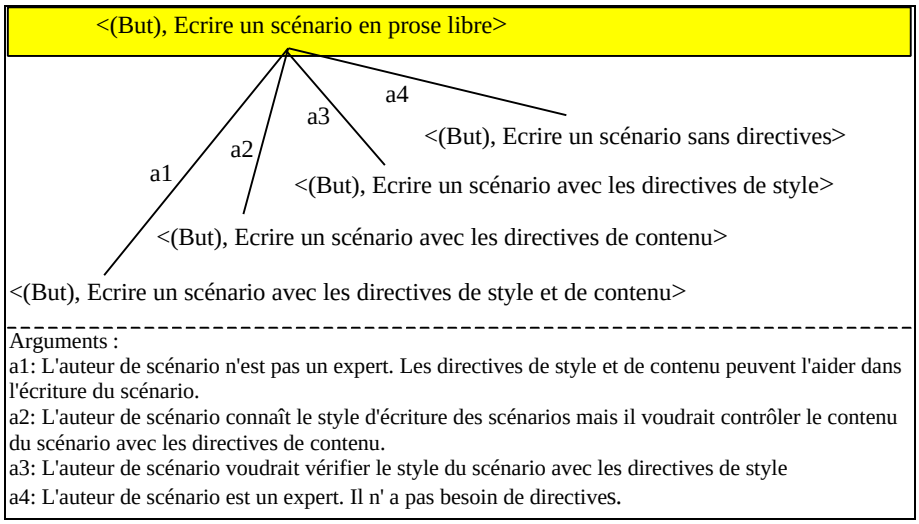


Figure 29 : Exemple de directive choix

La Figure 29 propose un exemple de directive choix. Dans cet exemple, l'intention *"Ecrire un scénario en prose libre"* est affinée par quatre alternatives proposant différentes manières pour satisfaire cette intention : (1) *"avec les directives de style et de contenu"*, (2) *"avec les directives de style"*, (3) *"avec les directives de contenu"* ou (4) *"sans directives"*. Chaque alternative est argumentée par des critères de choix qui aident l'ingénieur d'applications dans son choix de l'alternative la plus appropriée dans son cas. La première alternative est destinée à un auteur de scénario qui n'est pas un expert dans leur écriture. Des directives de style et de contenu sont alors proposées pour l'aider à accomplir sa tâche. Les deuxième et troisième alternatives sont destinées à un auteur de scénarios qui n'est pas novice dans ce domaine mais qui voudrait contrôler le contenu ou le style des scénarios. Finalement, la dernière alternative est destinée à un auteur de scénarios expert dans ce domaine et qui n'a pas besoin d'aide.

4.2.2 Directive plan

Une directive plan correspond à un problème complexe qui, pour être résolu, nécessite d'être décomposé en un ensemble de sous-problèmes. L'ingénieur d'applications connaît l'ensemble des décisions qui lui permettront d'atteindre le but qu'il poursuit. Il a un plan composé d'un ensemble de sous-directives. Le lien hiérarchique entre les directives est appelé le *lien de composition* à la Figure 7. Les directives composantes peuvent être des directives simples, tactiques ou stratégiques (Figure 28).

Dans la méthode CREWS-*L'Ecritoire* par exemple, pour atteindre l'intention *"Découvrir un but alternatif en utilisant le modèle de but"*, on doit répéter un certain nombre de fois les quatre décisions *"Réécrire le but suivant le modèle de but"*, *"Identifier des valeurs alternatives à un paramètre du but"*, *"Construire un nouveau but"* et *"Sélectionner un but"*. Comme le montre la Figure 30, ceci est modélisé par la directive plan *<(But), Découvrir un but alternatif en appliquant le modèle de but>* qui est composée de quatre directives *<(But avec état(But)=non structuré, Réécrire le but suivant le modèle de but>*, *<(But avec état(But)=structuré, Identifier des valeurs alternatives à un paramètre du but>*, *<({Nouvelle valeur de paramètre}), Construire un nouveau but>* et *<(But), Sélectionner un but)*.

L'ordre d'exécution des directives composants est défini dans le *graphe de précédence* (Figure 28). Il y a un graphe par directive plan. Les nœuds du graphe sont des directives (les composants du plan), alors que les arcs appelés *liens de précédence* représentent des transitions ordonnées ou parallèles entre directives.

Les *critères de choix* attachés aux liens permettent de prescrire les conditions d'occurrence d'une transition. Ils sont construits à partir d'arguments et sont de même nature que ceux attachés aux alternatives des contextes choix. Leur but est, ici, d'aider l'ingénieur d'applications à choisir le chemin à suivre pour l'exécution du plan.

Dans certains cas, le graphe de précédence peut être simple et ne proposer qu'un seul chemin pour exécuter le plan. Il est alors inutile d'associer des critères de choix aux liens de précédence. C'est le cas, par exemple, des plans dont les composants sont organisés en séquence.

Mais un graphe peut aussi contenir plusieurs chemins d'exécution. Ceci permet d'introduire plus de flexibilité dans la démarche capturée dans le composant.

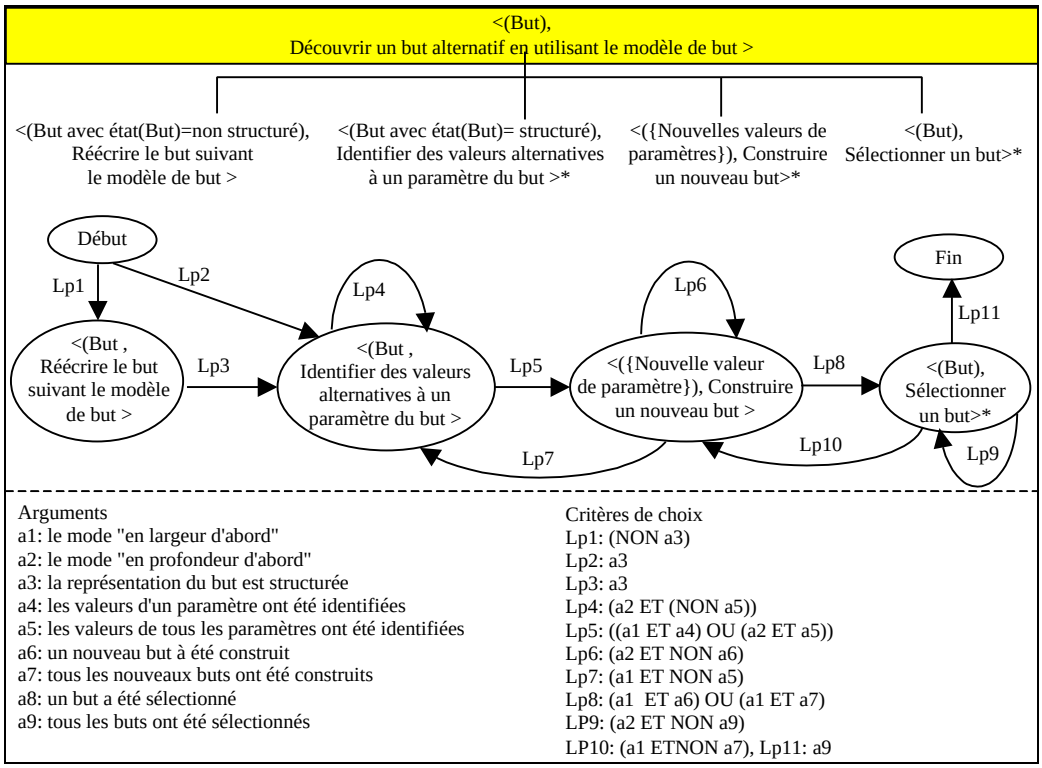


Figure 30 : Exemple de directive tactique de type plan

Exécuter un plan revient à parcourir son graphe de précédence. Lors de l'exécution d'un contexte plan, les critères de choix des liens de précédence sont calculés grâce à l'évaluation de leurs arguments.

4.2.3 Hiérarchie de directives

Les directives plan et choix sont définies récursivement au moyen de la notion de directive. Ce sont des agrégats composés d'autres directives sous forme de hiérarchies de directives. Les feuilles d'une

telle hiérarchie sont des directives simples. Par exemple, la directive ayant la signature *<(But avec état(But)=non structuré), Découvrir un but alternatif au but initial en utilisant le modèle de but>* a le modèle de processus défini sous forme d'une directive tactique. Comme le montre la Figure 31, le corps de la directive est représenté par une hiérarchie de directives tactiques et exécutables.

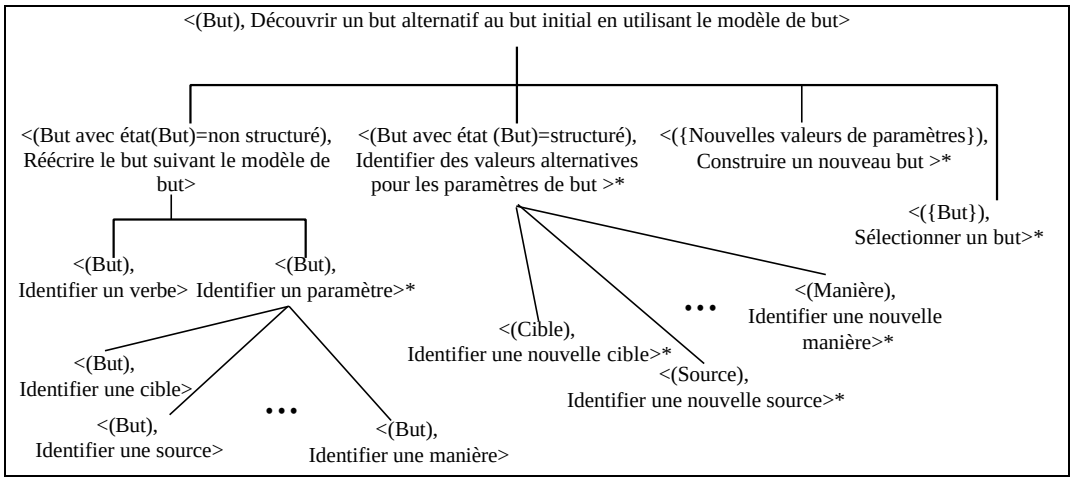


Figure 31 : Exemple d'une hiérarchie de directives tactiques

4.2.4 Directive tactique versus composant de méthode

Comme nous l'avons déjà mentionné à la section 2.2.2, une directive réutilisable indépendamment de sa méthode d'origine devient un composant de méthode. De plus, si cette directive est une directive tactique et que toutes ses sous-directives sont également des modules de processus réutilisables, elles peuvent aussi être représentées par des composants de méthode de niveau d'abstraction plus bas que le composant de départ. Ainsi, le composant de départ est un *composant agrégat*.

Pour qu'une directive tactique de type choix devienne un composant agrégat, il faut que toutes ses alternatives soit des directives réutilisables et représentées par des composants de méthode. Ainsi, le composant agrégat relie à travers le lien de choix un ensemble de sous-composants, un pour chaque directive alternative. Ces directives alternatives sont représentées par des composants à part entière. L'exécution d'un composant agrégat consiste à choisir l'un de ses sous-composants, le plus adapté à la situation donnée, et à l'exécuter. Tous ses sous-composants peuvent être à leur tour, atomiques ou agrégats.

Prenons comme exemple la directive exprimée à la Figure 29. Elle représente un fragment de processus réutilisable indépendamment de la méthode d'origine dans la construction d'autres méthodes. Elle est alors représentée par un composant de méthode qui est illustré à la Figure 32. De plus, c'est un composant agrégat car sa directive est une directive tactique de type choix, et toutes les alternatives dans ce choix sont des directives réutilisables indépendamment les unes des autres. Dans ce cas, chaque alternative est définie en tant que composant de méthode à part entière en lui associant les parties de produit nécessaires ainsi qu'un descripteur.

Dans la Figure 32 ainsi que dans les autres figures de cette section représentant des composants de méthode, la partie foncée représente le descripteur du composant, la partie grise clair sa signature et la partie blanche le corps de la directive et les éléments qui lui sont associés. La Figure 32 montre également les liens vers les sous-composants du composant agrégat.

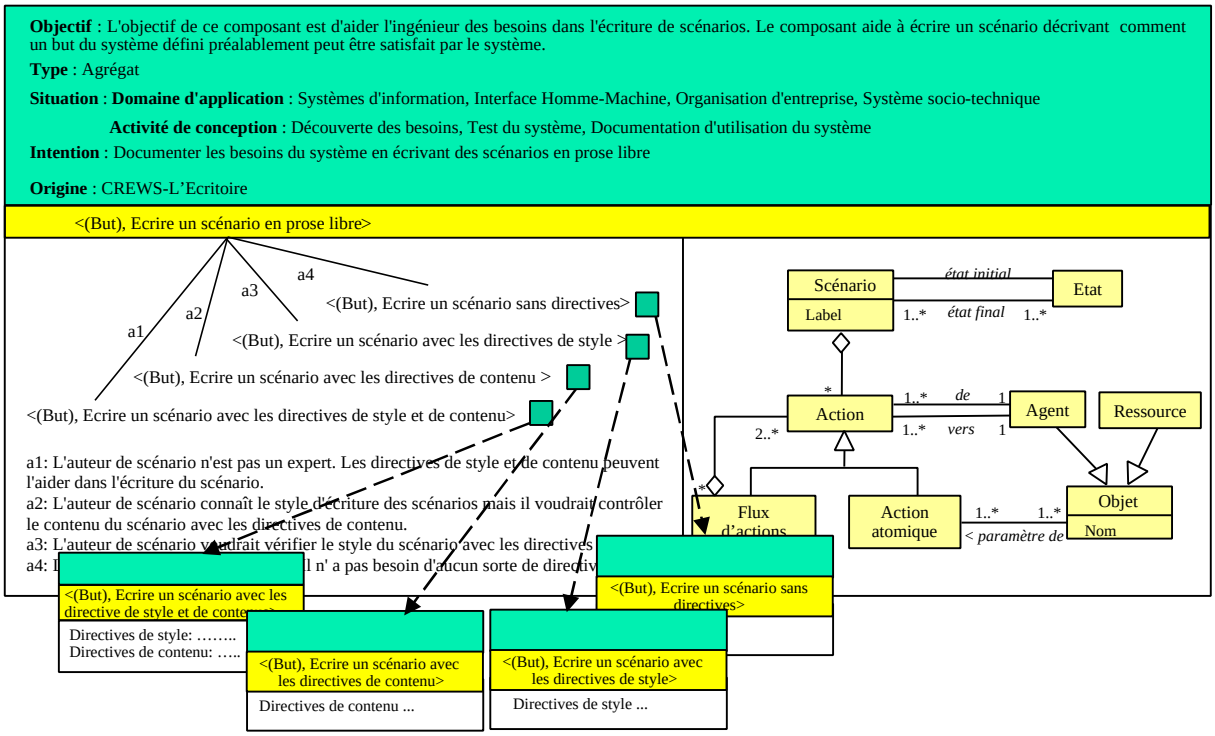


Figure 32 : Exemple de composant de méthode de type agrégat

De la même manière, la directive tactique de type plan peut relier à travers le lien de composition un ensemble de directives étant des directives réutilisables indépendamment. Dans ce cas, chaque directive du plan est représentée par un composant de méthode de niveau d'abstraction plus bas que le composant de départ. De plus, il peut être un composant atomique ou agrégat. L'exécution d'un composant agrégat consiste à exécuter tous ses sous-composants.

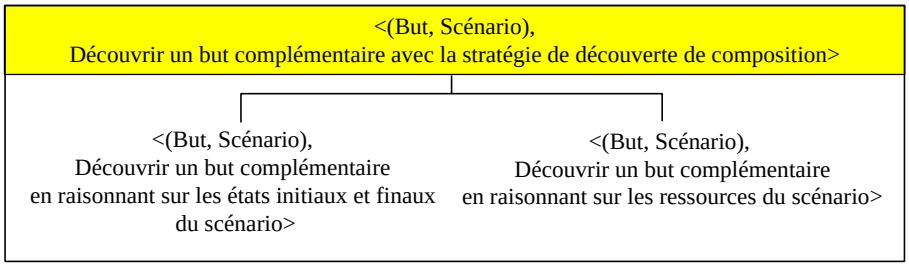


Figure 33 : Exemple de directive plan

Par exemple, la Figure 33 représente la directive d'un composant de méthode qui est exprimée sous forme d'un plan. Chaque élément de ce plan est aussi une directive qui peut être réutilisée indépendamment et, par conséquent, elle peut être représentée en tant que composant de méthodes à part entière. Le composant de la Figure 34 est alors un composant agrégat dont les éléments sont des composants atomiques.

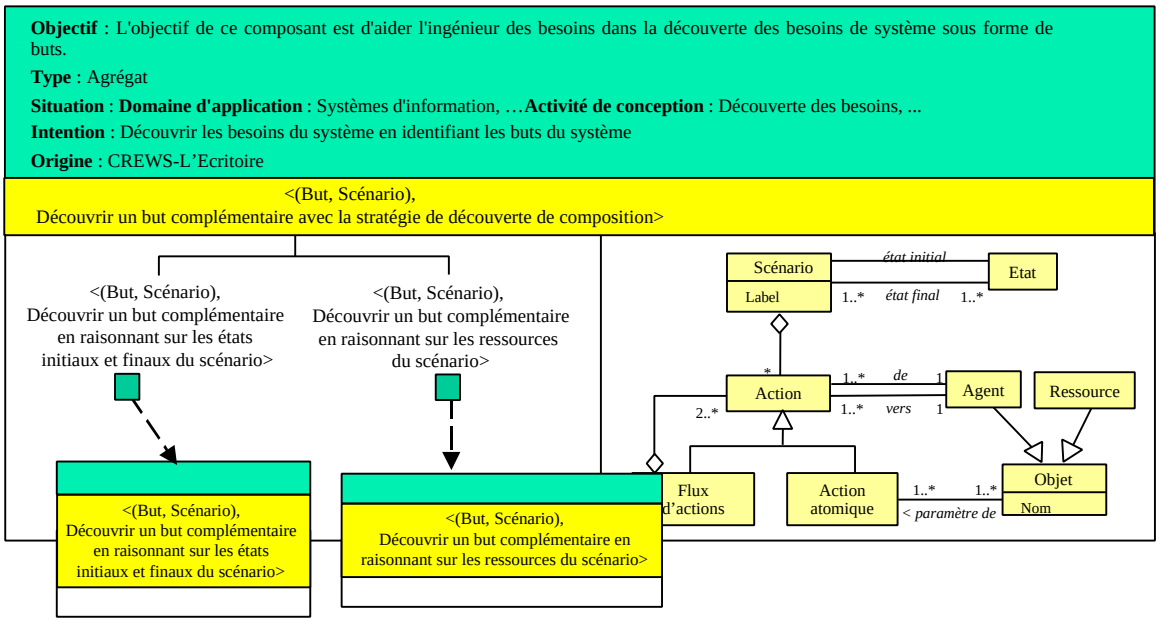


Figure 34 : Exemple de composant agrégat

Contrairement à ces deux cas, la directive tactique illustrée à la Figure 30 est un composant atomique car ses sous-directives ne sont pas réutilisables sans la directive qui les intègre.

4.3 Directive stratégique

Certaines méthodes proposent plusieurs manières possibles pour développer les produits correspondants. L'ingénieur d'applications a la possibilité de choisir une démarche parmi plusieurs suggérées par la méthode. Nous proposons d'utiliser la directive stratégique pour formaliser le modèle de processus d'une telle méthode.

Une directive stratégique permet de représenter un processus de développement *multi-démarches*. Un tel processus est représenté par un ensemble de directives qui sont reliées entre elles pour former un graphe de processus que l'on appelle une *carte*.

La carte propose une vue stratégique du processus en précisant ce qui peut être réalisé (quelle intention du processus) suivant quelle stratégie. Une directive stratégique est alors basée sur deux notions : l'*intention* et la *stratégie*. Le choix de ces deux concepts comme base du modèle de processus stratégique est fondé sur les deux hypothèses suivantes :

1. Nous pensons que le processus d'ingénierie des systèmes est orienté-intention. A tout moment, l'ingénieur d'applications est confronté à une intention, un objectif, qu'il désir à réaliser. Cette caractéristique est prise en compte par la carte en identifiant l'ensemble des intentions qui doivent ou peuvent être accomplies pour aboutir au produit escompté.

2. Il y a en général plusieurs stratégies pour réaliser une même intention. Par exemple, l'approche CREWS-*L'Ecritoire* propose plusieurs stratégies pour identifier des buts du système.

Nous représentons la structure de la directive stratégique à la Figure 35. Un couple d'intentions et une stratégie déterminant la manière de satisfaire l'intention cible à partir de l'intention source est appelé une *section*. Une carte est alors faite d'un ensemble de sections. A chaque section est associée une directive définissant comment réaliser l'intention cible à partir de l'intention source en appliquant la stratégie donnée. Cette directive est appelée une *directive de réalisation d'intention (DRI)*. Les sections dans la carte s'enchaînent les unes après les autres ou se présentent en parallèle entre deux intentions en construisant ainsi un graphe. Une directive stratégique est alors un graphe de DRI.

La carte a deux intentions particulières appelées *Démarrer* et *Arrêter*. Le nœud *Démarrer* permet de commencer la navigation dans la carte et le nœud *Arrêter* permet de terminer cette navigation. Il est évident qu'il y a de nombreux chemins possibles dans le graphe; chacun d'eux correspond à une démarche possible pour conduire les processus d'ingénierie. C'est donc en ce sens que la carte est un modèle *multi-démarches*.

Tout enchaînement des sections dans la carte est appelé une *route*. Puisque chaque section a une directive de réalisation d'intention associée, ces directives sont reliées en séquence par un *lien d'enchaînement* vu à la Figure 7. Cet enchaînement de DRI est aussi une directive associée à toute la route et appelée *directive de séquence* (Figure 35).

Les sections de la carte peuvent aussi se présenter en parallèle, c'est-à-dire, plusieurs sections peuvent relier la même intention source avec la même intention cible. Leurs DRI correspondantes sont reliées en parallèle par le *lien ET/OU* vue à la Figure 7. L'ensemble de ces directives construisent une nouvelle directive appelée une *directive de parallélisme* (Figure 35).

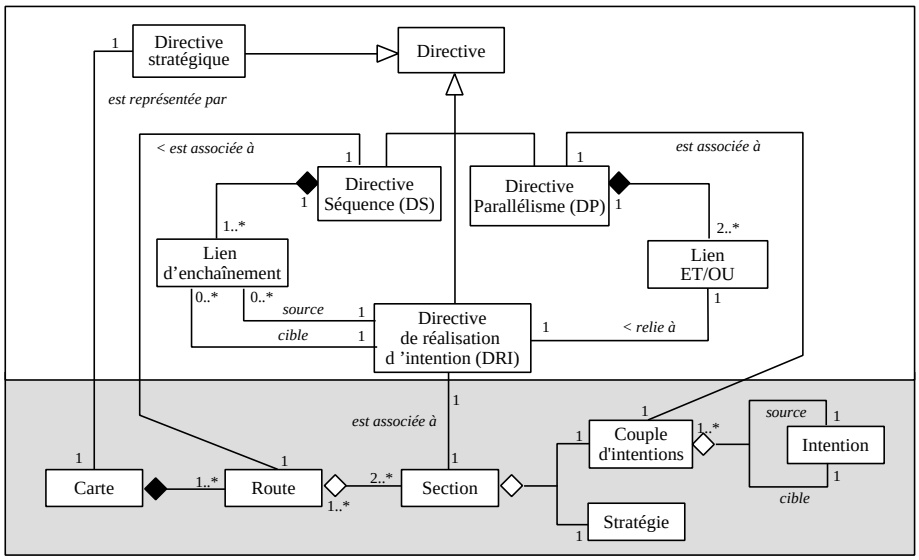


Figure 35 : Structure de directive stratégique

Autrement dit, une directive stratégique c'est un ensemble de DRI organisées sous la forme d'une carte. La carte est un moyen d'organisation des directives. L'ingénieur d'applications réalise les DRI suivant un chemin dans la carte. Pour bien comprendre la notion et la richesse d'une directive stratégique, nous présentons tout d'abord la structure sur laquelle elle est basée, c'est-à-dire la structure de carte et ensuite tous les types de sous-directives qu'elle comporte.

Une directive stratégique est un composant de méthode de type agrégat dans la plupart des cas. Toute DRI, DS ou DP faisant partie de la directive stratégique peut également être un composant de méthode de niveau de granularité plus fin.

4.3.1 Carte

Du point de vue structurel, une carte est un graphe directionnel dans lequel les nœuds sont des *intentions* nécessaires à la réalisation du processus d'ingénierie capturé dans la directive et les arcs entre les intentions sont des *stratégies* représentant les manières pour réaliser les intentions. Les stratégies connectent les intentions et permettent de progresser d'une intention vers une autre.

Afin de développer la notion de carte de manière plus explicite, considérons l'exemple de la directive issue de l'approche CREWS-*L'Ecritoire* dont l'intention est de découvrir les besoins fonctionnels d'un système d'information. C'est une directive stratégique représentée par la carte de la Figure 36. Dans cet exemple, le processus est basé sur la découverte des buts du système et l'écriture de scénarios. A chaque but on associe un scénario qui décrit le comportement du système dans la réalisation de ce but. La découverte des nouveaux buts alternatifs et complémentaires est basée sur l'analyse de la structure d'un but découvert préalablement et sur l'analyse de la structure et du contenu de son scénario associé. Le processus est décomposé en trois intentions d'ingénierie (*Découvrir un but, Ecrire un scénario* et *Conceptualiser un scénario*) et deux intentions spécifiques (*Démarrer* et *Arrêter*) permettant de commencer et d'arrêter le processus de développement. Huit stratégies connectent ces intentions, chacune définissant une manière particulière pour satisfaire l'intention cible.

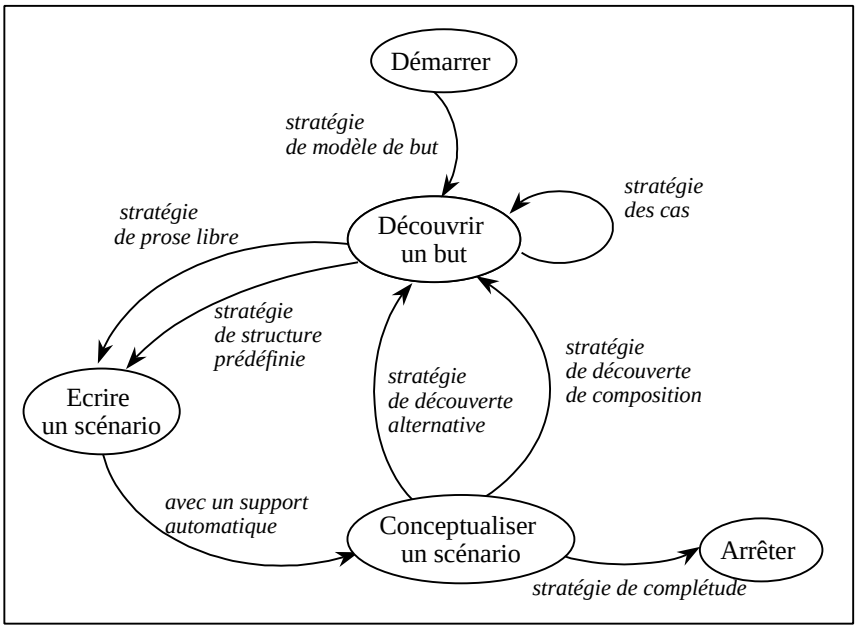


Figure 36: Exemple de carte

L'expression de l'intention de la carte suit la même structure que l'intention définie dans la signature des directives présentée à la section 3.2.2. Elle se limite à un *verbe* et à un paramètre *cible*. Les autres paramètres comme la source, la destination, la manière ou le moyen ne peuvent pas être précisés dans la carte puisqu'ils varient en fonction de la stratégie sélectionnée pour satisfaire l'intention.

L'ordonnancement d'intentions et de stratégies est non-déterministe. Une carte est une structure de navigation dans le sens où elle permet à l'ingénieur d'applications de déterminer un chemin d'intentions en partant de l'intention *Démarrer* pour finalement aboutir à l'intention *Arrêter*. Lorsque l'ingénieur d'applications utilise une directive stratégique, il progresse dans la carte en choisissant les intentions et les stratégies les plus appropriées à la situation du moment. Il construit son chemin de progression dans la carte en même temps qu'il applique la directive. Pour aider l'ingénieur d'applications à progresser dans la carte, un ensemble de directives de guidage est associé à celle-ci. Nous présentons ces directives à la section 4.3.5.

La carte contient un nombre fini de chemins, chacun d'eux prescrivant une façon de développer le produit, c'est à dire que chacun d'eux est un modèle de processus. Par conséquent, la carte est multi-modèle. Elle contient plusieurs modèles de processus, plusieurs démarches pour procéder à l'ingénierie des systèmes.

Comme le montre la partie grise de la Figure 35, l'élément principal de la carte est une *section*. De ce point de vue, une carte est un ensemble de sections correspondant chacune à un triplet $<I_i,I_j,S_{ij}>$ où I_i est une intention source, I_j est une intention cible et S_{ij} une stratégie permettant de réaliser l'intention I_j à partir de l'intention I_i. Par exemple, la carte présentée à la Figure 36 contient 8 sections. Une DRI est associée à chaque section. La carte est, donc, un modèle de processus défini par un ensemble d'intentions et un ensemble de stratégies, les stratégies permettant de naviguer d'intention en intention et de les réaliser en appliquant les DRI correspondantes.

Comme partie du triplet $<I, I, S>$, la stratégie caractérise la manière de progresser de l'intention I_i à l'intention I_j. La carte représente toutes les interconnexions entre intentions et stratégies. Formellement, I est un ensemble d'intentions, S est un ensemble de stratégies et la carte est un sous-ensemble du produit Cartésien $I \times I \times S$.

La manière spécifique d'accomplir une intention est capturée dans une section de la carte alors que les diverses sections qui ont la même intention I_i comme source et la même intention I_j comme cible définissent les différentes stratégies qui peuvent être adoptées pour accomplir l'intention I_j à partir de l'intention I_i. De la même façon, il peut y avoir différentes sections qui ont l'intention I_i comme source et les intentions I_{j1}, I_{j2},... I_{jn} comme cibles. Celles-ci montrent les différentes intentions qui peuvent être atteintes après la réalisation de l'intention I_i.

Les sections s'enchaînent de telle manière que l'intention cible de la première section est une intention source de la deuxième. Tout enchaînement des sections dans la carte est appelé une *route* (Figure 35). Une directive associée à une route est appelée une *directive de séquence*. Elle définit une suite possible de DRI. Par exemple dans la carte de la Figure 36, la section *<Découvrir un but, Ecrire un scénario, Stratégie de prose libre>* peut être suivie par la section *<Ecrire un scénario, Conceptualiser un scénario, Avec un support automatique>* ce qui veut dire que la suite des DRI correspondant à ces sections est une directive de *séquence*.

Deux intentions peuvent être connectées par plusieurs stratégies de même direction proposant des manières alternatives ou complémentaires pour satisfaire l'intention cible. En ce sens, la carte offre

des routes multi-progression pour réaliser une intention. Les DRI correspondant à ces sections peuvent être regroupées et présentées sous forme d'une directive de *parallélisme*.

Dans l'exemple présenté à la Figure 36, deux stratégies différentes sont proposées pour satisfaire l'intention *"Ecrire un scénario"* en partant de l'intention *"Découvrir un but"*: "*stratégie de prose libre"* et "*stratégie de structure prédéfinie"*. Il existe alors deux sections ayant l'intention *"Découvrir un but"* comme source et l'intention *"Ecrire un scénario"* comme cible. Les DRI de ces deux sections sont alternatives, car elles désignent deux manières différentes pour aboutir au même résultat. La DRI avec "*stratégie de prose libre"* suggère d'écrire un scénario librement en langage naturel, avec ou sans aide sur le contenu et le style du scénario tandis que la DRI avec la "*stratégie de structure prédéfinie"* propose une structure prédéfinie qui aide dans l'écriture d'un scénario. L'ingénieur d'applications choisit la manière la plus approprié à son goût. L'ensemble des deux DRI forme une directive parallélisme.

Dans le même exemple, il y a aussi deux stratégies pour progresser de l'intention *"Conceptualiser un scénario"* vers l'intention *"Découvrir un but"*: "*stratégie de découverte alternative"* et "*stratégie de découverte de composition"*. Contrairement au cas précédent, les DRI des deux sections respectives sont complémentaires. Dans le processus de la découverte des besoins fonctionnels du système la *"stratégie de découverte alternative"* permet de trouver tous les buts qui sont alternatifs à un but donné, c'est-à-dire toutes les variantes de fonctionnement du système possibles pour satisfaire un but, les cas échéants compris et la *"stratégie de découverte de composition"* permet d'identifier les buts complémentaires à celui de départ. Pour aboutir à un ensemble complet des besoins fonctionnels du système, l'ingénieur d'applications doit appliquer les deux stratégies plusieurs fois, mais l'ordre de leur application n'est pas important. L'ensemble des deux DRI est aussi une directive parallélisme.

La carte peut aussi avoir des stratégies réflexives. C'est le cas de la *"stratégie des cas"* qui boucle sur l'intention *"Découvrir un but"* dans l'exemple de la Figure 36. Suivant cette stratégie, de nouveaux buts peuvent être découverts à partir d'un but trouvé préalablement. Ces nouveaux buts sont alternatifs à celui de départ.

4.3.2 Directive de réalisation d'intention

Une *Directive de Réalisation d'Intention* (*DRI*) aide à la réalisation d'une intention selon la stratégie donnée. Pour chaque section de la carte il existe une DRI. Elle fournit le moyen opérationnel de satisfaire l'intention cible de la section.

Par souci d'homogénéisation la signature d'une DRI associée à une section $<I_i, I_j, S_{ij1}>$ de la carte est un couple <situation, intention> fait comme suit :

- la situation comporte la partie de produit qui est la cible de l'intention I_i et dont l'état peut être précisé par une condition d'occurrence,
- l'intention est exprimée sous la forme I_j ***avec*** $S_{ij1.}$

Prenons comme exemple la DRI associée à la section <*Découvrir un but, Ecrire un scénario, Stratégie de prose libre*> de la carte représentée à la Figure 36. La signature de cette DRI est la suivante :

<(But), Ecrire $_{verbe}$ *(un scénario)* $_{cible}$ *(avec la stratégie de prose libre)* $_{manière}$

La situation de cette signature comporte la partie de produit *"But"* qui est la cible de l'intention source *"Découvrir* $_{verbe}$ *(un but)*$_{cible}$*"* de cette section. L'intention de la signature est composée de l'intention *"Ecrire un scénario"* qui est la source dans la section et de la stratégie *"de prose libre"* qui est exprimée à l'aide du paramètre *manière*.

Toute DRI est une directive de l'un des trois types possibles, c'est-à-dire simple, tactique ou directive stratégique. La DRI de notre exemple est illustrée à la Figure 29 en tant que directive tactique de type choix.

4.3.2.1 DRI versus composant de méthode

Dans la plupart des cas, la DRI associée à une section de la carte d'une méthode est vue comme un module réutilisable indépendamment de la carte à laquelle elle est associée. Une telle DRI est représentée comme un composant de méthode de granularité plus fine que le composant ayant comme directive la carte initiale. La DRI et ses parties de produit associées devient un composant de méthode lorsqu'on lui a associé un descripteur. Dans ce cas, le composant ayant comme directive la carte initiale est un composant agrégat et le nouveau composant est soit un composant atomique soit lui même un agrégat. Par conséquent, la DRI peut être réutilisée et assemblée avec d'autres composants en tant que composant à part entière ou bien en tant que partie d'un composant agrégat.

La DRI de l'exemple ci-dessus, représenté à la Figure 29, est une directive réutilisable indépendamment de sa méthode d'origine. Elle est, de ce fait, définie comme un composant de méthode que nous illustrons à la Figure 32.

Nous pouvons aller plus loin dans la démonstration de la décomposition des directives et l'existence de composants de différents niveaux d'abstraction. La directive du même exemple (Figure 32) est de type tactique, représentée sous forme d'un choix d'alternatives. De plus, chaque directive alternative de ce choix peut être réutilisée indépendamment et par conséquent peut être définie en tant que composant de méthode à part entière. Par conséquent, le composant de méthode de la Figure 32 est aussi un composant agrégat.

La DRI associée à la section <*Découvrir un but, Découvrir un but, Stratégie de modèle de but*> de la même carte, au contraire, est définie comme un composant de méthode atomique que nous illustrons à la Figure 37. Même si la structure de la DRI est complexe (c'est une directive tactique de type plan) le composant correspondant à cette directive reste atomique, car ses sous-directives ne sont pas réutilisables sans celle qui les intègre.

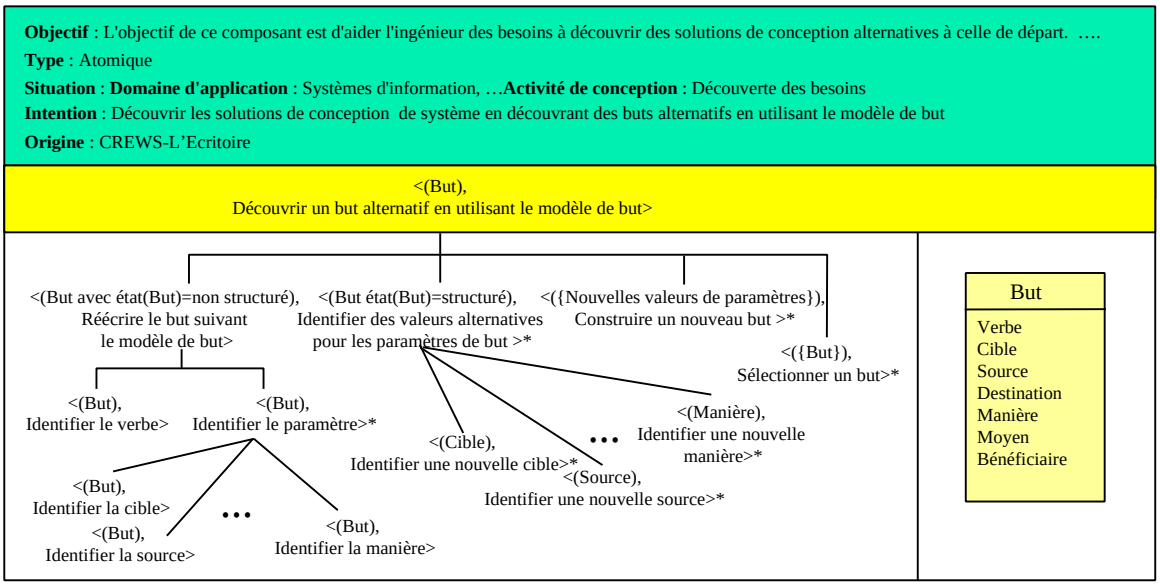


Figure 37 : Exemple de composant atomique

Une DRI associée à une section d'une carte peut même être représentée par une directive stratégique, c'est-à-dire par une autre carte avec des directives associées.

4.3.3 Directive parallélisme

Une *Directive Parallélisme* (*DP*) imbrique plusieurs DRI correspondant aux sections parallèles d'une carte et construit ainsi une nouvelle directive de même niveau d'abstraction mais de niveau de granularité plus élevé. Autrement dit, pour deux intentions d'un carte, connectées par plusieurs stratégies de même direction, il existe une DP. Pour deux intentions I_i et I_j données la signature d'une DP est exprimée de la manière suivante:

- la situation comporte la partie de produit qui est la cible de l'intention I_i et dont l'état peut être précisé par une condition d'occurrence,
- l'intention est exprimée sous la forme I_j (sans préciser la stratégie).

Par exemple, dans la carte de la Figure 36, il y a deux sections ayant comme source l'intention *"Découvrir un but"* et comme cible l'intention *"Ecrire un scénario"*. Une DP regroupe les DRI de ces sections pour former une nouvelle directive. La signature de cette directive est exprimée sous la forme : *"<(But), Ecrire un scénario>"*. Le contenu de la DP est illustré à la Figure 38.

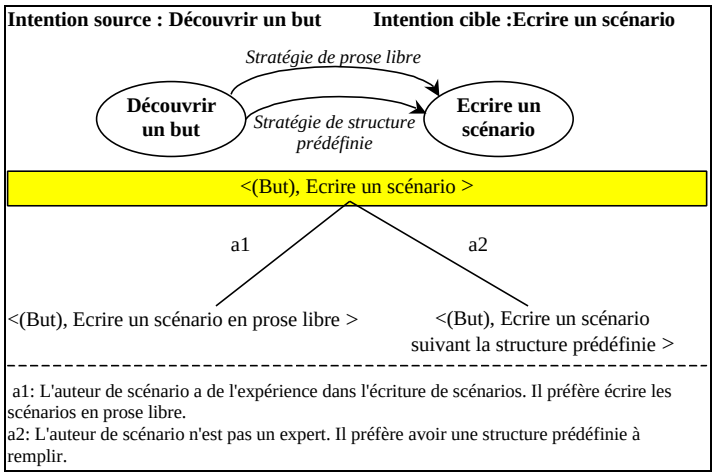


Figure 38 : Exemple de directive parallélisme

Une DP est une directive tactique. Les DRI imbriquées dans une DP peuvent être exclusives ou complémentaires. C'est la directive parallélisme qui définit quel est le rapport entre les directives qu'elle comporte. Dans notre exemple, les deux DRI imbriquées dans la DP sont exclusives. La DP contient des arguments qui aident l'ingénieur d'applications à choisir une des deux DRI.

4.3.3.1 DP versus composant de méthode

Si toutes les directives faisant partie d'une directive parallélisme sont définies en termes de composants de méthodes, la directive parallélisme est aussi un composant agrégat. Le composant agrégat est de même niveau d'abstraction que ses sous-composants. Dans notre exemple, la DRI de chaque section est déjà représentée par un composant de méthode. Par conséquent, la directive parallélisme qui les imbrique est aussi représentée par un composant de méthode qui est un composant agrégat.

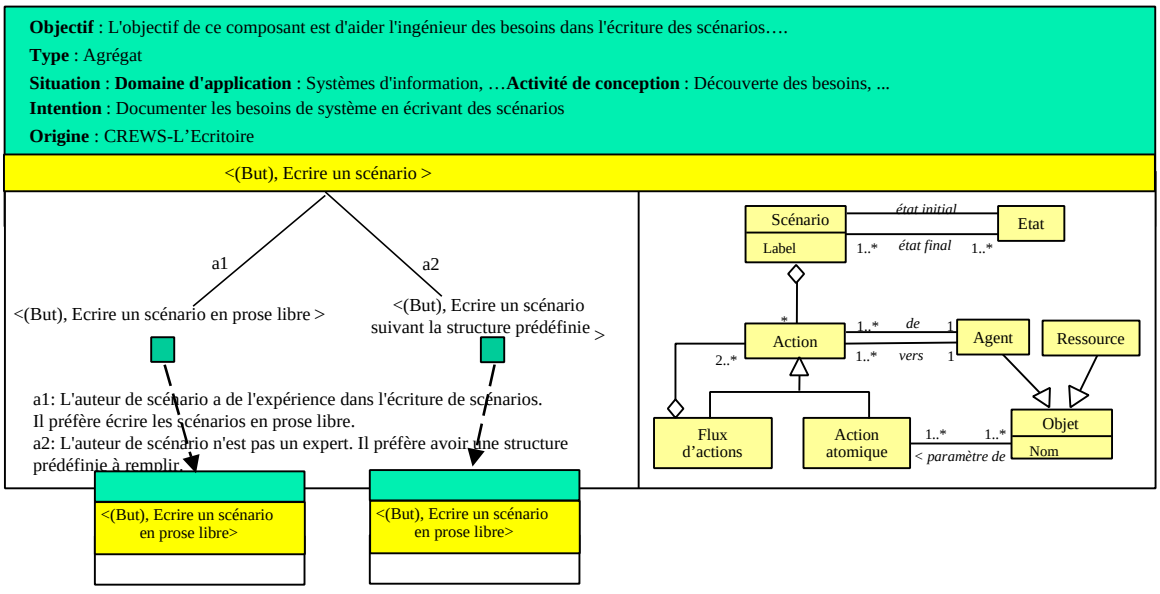


Figure 39 : Exemple de composant agrégat

Dans la même carte (Figure 36), une autre DP concerne les deux sections ayant comme source l'intention *"Conceptualiser un scénario"* et comme cible l'intention *"Découvrir un but"*. Dans ce cas les DRI de ces sections sont complémentaires. La directive parallélisme comportant ces deux DRI permet à l'ingénieur de développement d'exécuter les deux DRI ou seulement l'une d'entre elles. Comme dans le cas précédent, la DP est représentée par un composant de méthode (de type agrégat) car les deux DRI imbriquées sont aussi des composants.

4.3.4 Directive séquence

Une *Directive Séquence* (*DS*) imbrique deux DRI correspondant aux sections d'une carte qui s'enchaînent et construit ainsi une nouvelle directive de même niveau d'abstraction mais de niveau de granularité plus élevé. Autrement dit, pour deux sections d'une carte où l'intention cible d'une section est l'intention source de la deuxième section, il existe une DS. Pour deux sections $<I_i, I_j, S_{ij1}>$ et $<I_j, I_k, S_{jk1}>$ d'une carte, la signature de la DP correspondante est exprimée de la manière suivante:

- la situation comporte la partie de produit qui est la cible de l'intention I_i et dont l'état peut être précisé par une condition d'occurrence,
- l'intention est exprimée sous la forme I_k (sans préciser la stratégie).

Dans la carte de la Figure 36, l'enchaînement de la section *<Découvrir un but, Ecrire un scénario, Stratégie de prose libre>* par la section *<Ecrire un scénario, Conceptualiser un scénario, Stratégie avec un support automatique>* a une DS dont la signature est exprimée de la manière suivante : *<(But), Conceptualiser un scénario>*.

La situation de cette signature comporte la partie de produit *"But"* qui est la cible de l'intention source *"Découvrir* $_{verbe}$ *(un but)*$_{cible}$*"* de la première section. L'intention de la signature correspond à l'intention *"Conceptualiser un scénario"* qui est la source dans la deuxième section.

Une DS est une directive tactique de type plan, car elle définit la séquence de deux directives. La Figure 40 illustre la directive de notre exemple.

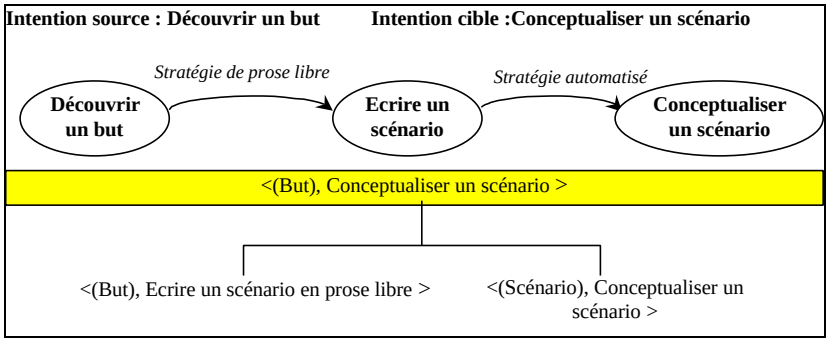


Figure 40 : Exemple de directive séquence

4.3.4.1 DS versus composant de méthode

Si toutes les DRI imbriquées dans une DS sont représentées sous forme de composants de méthode, la DS est aussi un composant de méthode mais de niveau de granularité plus élevé que les composants imbriqués. Dans notre exemple, la DS peut être vue comme un composant agrégat qui propose d'enchaîner le composant de l'écriture d'un scénario avec un composant de la conceptualisation d'un scénario (Figure 40).

4.3.5 Directives de progression associées à la carte

Dans un processus dirigé par les intentions, comme on le suppose dans une directive stratégique, l'ingénieur d'applications est confronté de manière répétitive aux deux questions suivantes :

(1) comment réaliser l'intention sélectionnée, et

(2) comment sélectionner la prochaine intention afin de progresser dans le processus.

Les directives de réalisation d'intention répondent à la première question (section 4.3.2). Pour répondre à la deuxième question, nous proposons d'associer à la carte de la directive stratégique un nouveau type de directives que nous appelons des *directives de progression*.

Etant donné que la directive stratégique propose plusieurs chemins possibles pour satisfaire l'intention de départ, les directives de progression aident l'ingénieur d'applications dans le choix de ce chemin. A tout moment, elles aident cet ingénieur à sélectionner les intentions et les stratégies suivantes pour progresser dans la carte. En sélectionnant l'intention d'arrivée et une stratégie pour y aller, l'ingénieur d'applications sélectionne une DRI qui est associée à la section choisie et la réalise. En conséquence, il construit le processus de développement dynamiquement.

Nous distinguons deux types de directive de progression : les *directives de sélection de stratégie* et les *directives de sélection d'intention*. La Figure 41 montre comment les directives de progression sont associées à la carte. Nous considérons dans cette section les différentes directives de progression et leurs relations avec la carte, nous détaillons également la structure et le contenu de ces directives.

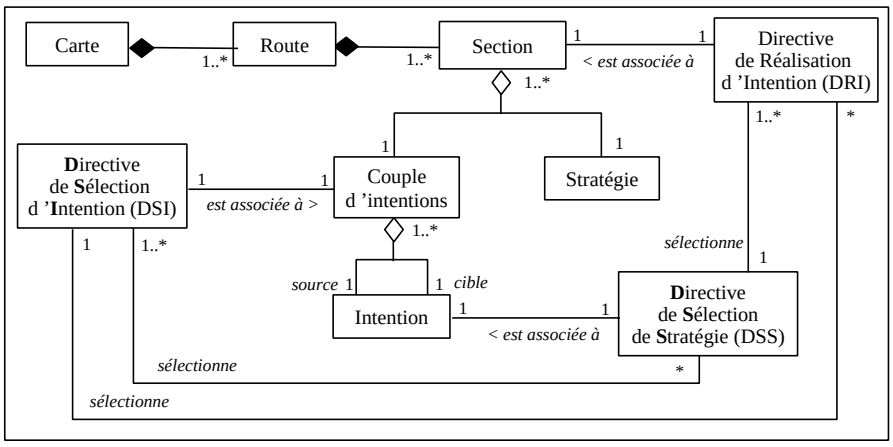


Figure 41 : Les directives associées à la carte

4.3.5.1 Directive de sélection de stratégie

Une *Directive de Sélection de Stratégie* (*DSS*) détermine quelles sont les stratégies connectant deux intentions et aide à choisir l'une d'entre elles. Elle peut être appliquée lorsque l'intention source et l'intention cible sont déterminées et qu'il y a plusieurs stratégies possibles pour satisfaire l'intention cible à partir de l'intention source. Le rôle de la DSS est de guider la sélection de la stratégie la mieux appropriée pour la situation donnée.

Autrement dit, pour un couple d'intentions connectées par plus d'une stratégie de même direction, il existe une DSS.

Comme toute directive, la DSS est décrite en utilisant le méta-modèle présenté à la Figure 7. Rappelons que chaque directive a une signature qui est définie par un couple <situation, intention> où la situation comporte une ou plusieurs parties de produit en cours de construction et l'intention dit quel est l'objectif que la directive souhaite atteindre (section 3.2).

Le rôle de la DSS est de guider la sélection de la stratégie la mieux appropriée pour la situation donnée. Elle peut être appliquée lorsque l'intention source et l'intention cible sont déterminées et qu'il y a plusieurs stratégies possibles pour satisfaire l'intention cible à partir de l'intention source. Etant donnée la spécificité de cette directive, nous spécialisons sa signature. La signature d'une DSS associée à un couple d'intentions $<I_i, I_j>$ est exprimée de la manière suivante :

- la situation comporte la partie de produit qui est la cible de l'intention I_i ; une condition d'occurrence peut préciser l'état de cette partie de produit,
- l'intention est exprimée sous la forme : ***Progresser*** $_{verbe}$ ***(vers*** I_j***)*** $_{cible}$.

Le verbe *Progresser* est toujours utilisé pour exprimer les intentions des DSS d'une manière uniforme. De plus, le mot *vers* précise quelle est l'intention cible de la progression.

Prenons comme exemple la carte de la méthode CREWS-*L'Ecritoire* présentée à la Figure 36. Elle propose deux sections pour progresser à partir de l'intention *"Découvrir un but"* vers l'intention *"Ecrire un scénario"*. Une DSS est associée à ce couple d'intentions. La signature de cette DSS est la suivante : *"**Progresser*** $_{verbe}$ ***(vers** Ecrire un scénario)* $_{cible}$*"*.

Une DSS est toujours une directive tactique. Elle est composée de directives exécutables, une par section à sélectionner. Chacune de ces directives exécutables contient une action de délégation qui sélectionne une DRI associée à la section. Une DSS ne peut pas être représentée par un composant de méthode car elle ne peut pas être réutilisée sans la carte à laquelle elle est associée.

Dans notre exemple, la directive de sélection de stratégie illustrée à la Figure 42 est une directive tactique de type choix. Elle aide l'ingénieur d'applications à faire son choix parmi les deux sections et à progresser ainsi dans la carte. Les deux alternatives dans cette DSS sont des directives exécutables dont les actions sont de type délégation. Elles délèguent le processus d'ingénierie à d'autres directives, dans notre cas aux DRI associées aux sections correspondantes. En réalité, l'ingénieur d'applications

sélectionne une DRI associée à la section grâce à l'action de délégation de la directive exécutable et l'exécute.

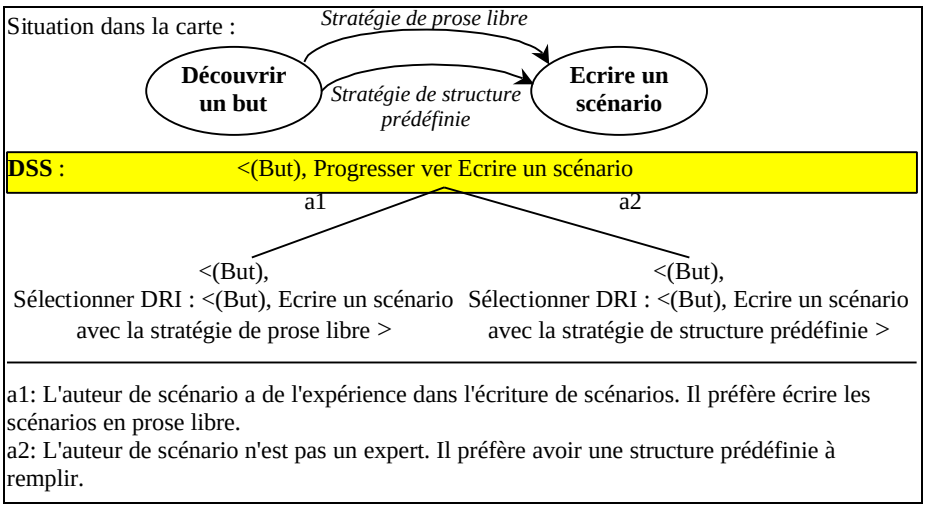


Figure 42 : Exemple de DSS

4.3.5.2 Directive de sélection d'intention

Une *Directive de Sélection d'Intention* (*DSI*) détermine quelles sont les intentions qui peuvent succéder à l'intention donnée et aide à choisir l'une d'entre elles. Elle peut être appliquée lorsqu'une intention vient d'être réalisée et que l'ingénieur d'applications doit déterminer quelle sera l'intention à réaliser à la prochaine étape. Puisque plusieurs intentions peuvent être accomplies dans la prochaine étape, le rôle de la DSI est de guider la sélection de l'intention suivante et de fournir l'ensemble des DRI et DSS correspondantes.

La signature d'une DSI associée à une intention I_i de la carte est exprimée de la manière suivante:

- la situation comporte la partie de produit qui est la cible de l'intention I_i ; elle peut être précisée par une condition d'occurrence,
- l'intention est exprimée sous la forme : ***Progresser*** $_{verbe}$ ***(de*** I_i***)*** $_{source}$

L'intention de la signature de la DSI exprime le fait que la directive aide l'ingénieur d'applications à progresser dans la carte. Le verbe *"Progresser"* est utilisé à ce propos. Le paramètre *source* exprime l'intention qui est la source de progression.

Prenons comme exemple toujours la même carte présentée à la Figure 36. Pour progresser à partir de l'intention *"Découvrir un but"*, l'ingénieur d'applications a deux possibilités : soit aller vers l'intention *"Ecrire un scénario"*, soit boucler sur la même intention. Une DSI est associée à l'intention *"Découvrir un but"* . La signature de cette DSI est *"**Progresser*** $_{verbe}$ ***(de** Découvrir un but)* $_{source}$*"*.

Le corps d'une DSI définit quelles sont les intentions cibles de la progression et aide à choisir l'une d'entre elles. Une DSI est toujours une directive tactique. Comme dans le cas de la DSS, la DSI est composée d'un ensemble de directives exécutables comportant des actions de délégation. Si la

progression vers une intention cible est possible par plusieurs chemins, c'est-à-dire s'il y a plusieurs sections entre l'intention source et l'intention cible, l'action sélectionne la DSS qui correspond à cette situation. S'il n'existe qu'une seule section entre ces deux intentions, l'action sélectionne directement la DRI lui correspondant.

La Figure 43 illustre la DSI de notre exemple. C'est une directive tactique de type choix qui propose deux alternatives : progresser à partir de l'intention *"Découvrir un but"* vers l'intention *"Ecrire un scénario"* ou revenir sur l'intention *"Découvrir un but"* de manière réflexive. Les argument proposés par la directive guide l'ingénieur d'applications à faire son choix. Chacune des deux alternatives est une directive exécutable comportant une action de délégation. L'action de la première alternative sélectionne la DSS qui aide à progresser vers l'intention *"Ecrire un scénario"*, car il existe deux possibilités pour le faire (Figure 42). La deuxième alternative sélectionne la DRI associée à la section qui est l'unique section réflexive.

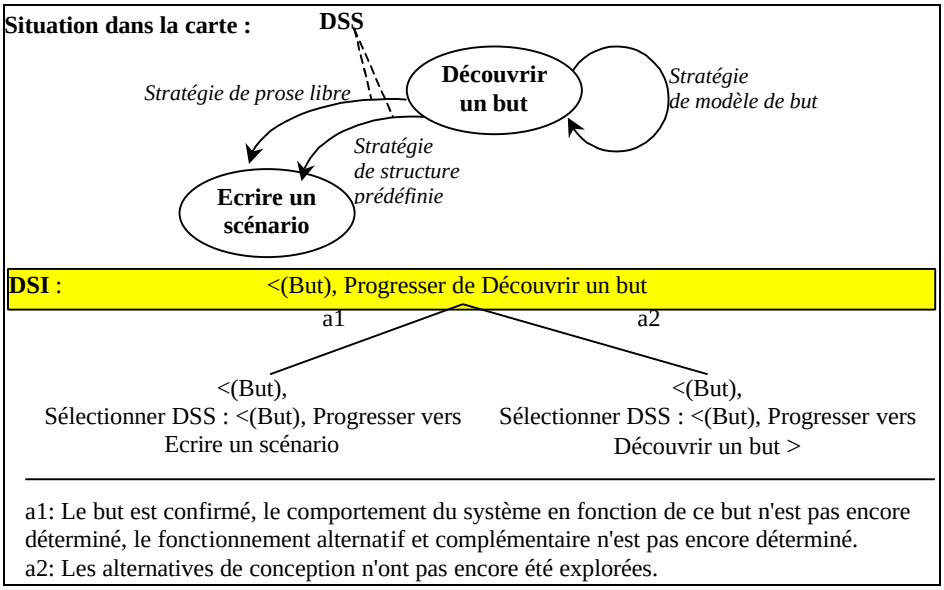


Figure 43 : Exemple de DSI

Une DSI ne peut pas être représentée par un composant de méthode car elle ne peut pas être réutilisée sans la carte à laquelle elle est associée.

5. CONCLUSION

Le méta-modèle de méthodes présenté dans ce chapitre permet de représenter toute méthode ou approche d'ingénierie sous forme d'assemblage de composants réutilisables. Ces composants peuvent être réutilisés dans la construction des nouvelles méthodes ou pour enrichir des méthodes existantes.

Notre méta-modèle met en avant la notion de processus que la méthode propose par rapport à celle de produit qu'elle permet de construire. Il est essentiellement ciblé sur la notion de directive qui est l'élément de base du modèle de processus. La richesse de la structure de directive que nous proposons dans ce mémoire permet de représenter n'importe quel type de démarche en commençant par la plus

simple – une instruction en langage naturel, jusqu'au modèle complexe de processus multi-démarches ayant une structure de graphe. Par conséquent, toute méthode existant peut être redéfinie en termes de composants suivant notre méta-model. Pour aider l'ingénieur de méthodes à réaliser cette tâche nous proposons un modèle de processus de re-ingénierie dans le chapitre suivant.

CHAPITRE 4

Modèle de processus de ré-ingénierie de méthodes sous forme des composants réutilisables

Dans le chapitre précédent nous avons présenté un méta-modèle de méthodes permettant de définir une méthode à base de composants réutilisables. Mais ce méta-modèle ne dit pas à l'ingénieur de méthodes comment redéfinir une méthode existante afin qu'elle devienne un assemblage de composants. Ce chapitre est consacré à la définition du modèle de processus de ré-ingénierie de méthodes existantes. Il s'agit, en fait, d'un modèle de processus qui aide l'ingénieur de méthodes à redéfinir toute méthode sous forme des composants en respectant le méta-modèle de méthodes présenté au Chapitre 3. Ces composants peuvent être stockés ensuite dans une base de méthodes et réutilisés dans la construction d'autres méthodes.

Le chapitre est composé de quatre sections. La première introduit notre modèle de processus de ré-ingénierie de méthodes développé ensuite dans la section 2. Dans la section 3 nous proposons un exemple d'application qui reconstruit le modèle des cas d'utilisation de la méthode OOSE en assemblage de composants de méthode. Finalement, nous concluons le chapitre par la section 4.

1. INTRODUCTION

Le terme *processus* dans le domaine de l'ingénierie de méthodes est employé, dans la majorité des cas, dans le sens d'"un ensemble d'activités inter-reliées et menées dans le but de définir un produit" [Franckson 91]. Dans notre travail nous nous intéressons aux différents processus de redéfinition de méthodes existantes dans le but de présenter ces méthodes sous forme de composants réutilisables.

Tout processus, autrement dit démarche, est une instance d'un modèle de processus. Un *modèle de processus* est l'abstraction d'une classe de processus. Il permet la représentation des caractéristiques communes aux processus d'une même classe. Dans la communauté des systèmes d'information, un modèle de processus correspond à la représentation méthodologique prescrite par la méthode utilisée. Dans notre cas, il s'agit, d'un modèle de processus qui est une abstraction de plusieurs démarches de reconstruction de méthodes et qui guide l'ingénieur de méthodes dans le choix de la démarche particulière. Ce modèle de processus est basé sur le méta-modèle de méthodes présenté au Chapitre 3.

Un *méta-modèle de processus* est la description d'un ensemble de modèles de processus. Notre modèle de processus utilise le méta-modèle de processus multi-démarches que nous utilisons pour décrire les modèles de processus des méthodes et que nous présentons au Chapitre 3. Ce méta-modèle utilise le concept de directive stratégique définie comme une carte de processus et de directives associées.

La Figure 44 illustre l'utilisation de ce modèle de processus par l'ingénieur de méthodes.

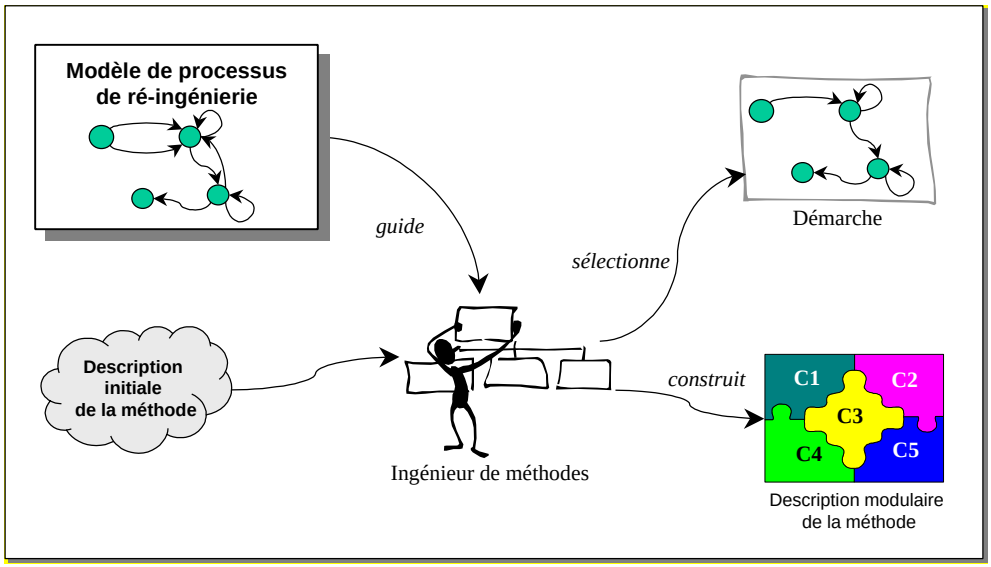


Figure 44 : Application du modèle de processus de ré-ingénierie des méthodes

La plupart de méthodes existantes sont décrites d'une manière informelle. Rares sont les méthodes proposant une description structurée du processus à réaliser pour construire le produit de la méthode. Toutefois, notre modèle de processus de ré-ingénierie prend en compte le fait que les méthodes peuvent avoir des représentations différentes. Comme le montre la Figure 44, l'ingénieur de méthodes sélectionne une démarche de reconstruction parmi plusieurs démarches proposées par le modèle de processus de ré-ingénierie en suivant le guidage proposé par celui-ci. En appliquant cette démarche il reconstruit la méthode choisie par assemblage de composants de méthode.

2. MODELE DE PROCESSUS DE RE-INGENIERIE DE METHODES

Le processus de ré-ingénierie de méthodes que nous proposons dans ce chapitre est basé sur la décomposition du modèle de processus de la méthode modélisée préalablement par une carte en directives réutilisables indépendamment de la méthode elle-même. Chacune de telles directives est définie ensuite comme un composant de méthode en lui associant des parties de produit nécessaires à son exécution ainsi qu'un descripteur.

Nous supposons ici que le modèle de processus de toute méthode peut être représentée par une directive stratégique, c'est-à-dire par une carte avec des directives de réalisation et de progression associées.

2.1 Carte de processus de ré-ingénierie

La Figure 45 illustre notre modèle de processus de ré-ingénierie de méthodes.

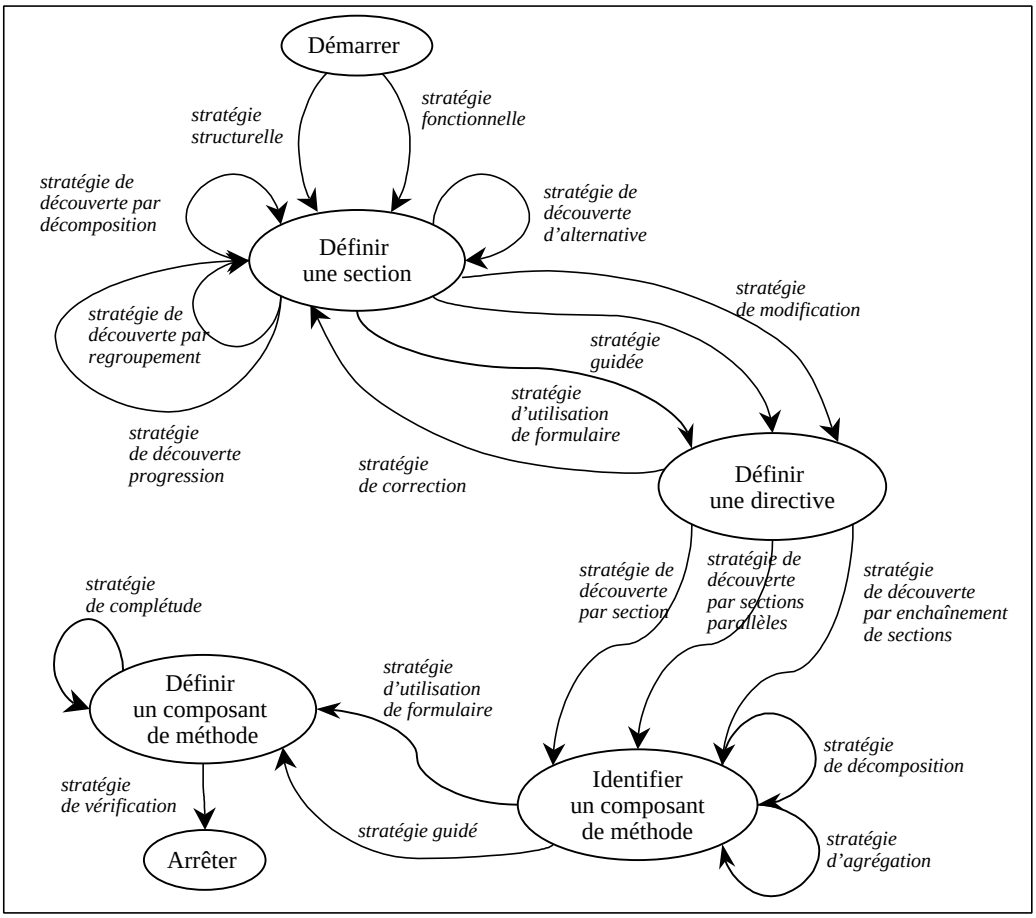


Figure 45 : Modèle de processus de ré-ingénierie de méthodes

Le processus de ré-ingénierie est décrit à l'aide d'une directive stratégique, c'est-à-dire à l'aide d'une carte et d'un ensemble de directives. La carte de ce processus contient les différentes intentions et stratégies pour réaliser l'intention *Reconstruire une méthode sous forme des composants réutilisables*.

Comme toute directive, notre processus de ré-ingénierie a pour signature <*(Représentation initiale d'une méthode M), Reconstruire la méthode M sous forme des composants réutilisables*>. Elle permet de redéfinir une méthode existante sous forme des composants.

Le processus de ré-ingénierie est basé sur la réalisation de quatre intentions : *Définir une section, Définir une directive, Identifier un composant de méthode* et *Définir un composant de méthode*. Le choix de ces intentions est basé sur notre conviction que toute méthode peut avoir un modèle de processus représenté par une directive stratégique qui est un modèle de processus multi-démarches composé de sections et des directives de réalisation d'intentions et de progression dans la carte associée. La définition des composants de méthode à son tour est basée sur la décomposition de cette carte en sections simples, parallèles ou enchaînements de sections.

Par conséquent, nous suggérons de redéfinir d'abord le modèle de processus de la méthode concernée sous forme d'une directive stratégique. Ceci est représenté dans notre carte de ré-ingénierie de méthodes par deux intentions :

- *Définir une section* qui aide l'ingénieur de méthodes à définir une section de la carte de la méthode.

- *Définir une directive* qui aide l'ingénieur de méthodes à définir la directive associée à une ou plusieurs sections. Elle aide à définir une étape de la carte contenant une intention source, une intention cible et une stratégie. Tandis que l'intention *Définir une directive* aide à définir la structure d'une directive qui peut être simple, tactique ou même stratégique (Chapitre 3).

Ces deux intentions permettent à l'ingénieur de méthodes de redéfinir le modèle de processus initial de la méthode (si elle en a un) ou même de le définir (si elle n'en a pas) sous forme d'une directive stratégique.

La deuxième partie de notre processus de ré-ingénierie concerne la décomposition du nouveau modèle de processus de la méthode concernée en sous-directives afin de les présenter en termes des composants. Selon notre définition de composant (Chapitre3), toute section dans la carte d'une méthode est un composant sinon la méthode est un composant atomique. Il est possible alors, de passer à l'identification des composants dès qu'une directive associée à une section est définie. Toutefois, nous sommes convaincus que la carte de la méthode doit être finalisée avant de commencer sa décomposition en composants de méthode. Dans le cas contraire, le processus permettant de décider si une directive est un composant ou non dès son identification pourrait augmenter le nombre de modifications à réaliser, car chaque fois qu'il nous faudrait effectuer des modifications sur les sections ou les directives de la carte qui n'est pas finalisée, on devrait également faire des modifications sur les composants concernés pas ces modifications.

Les deux intentions concernant la décomposition de la méthode en composants sont les suivantes :

- *Identifier un composant de méthode* aide l'ingénieur de méthodes à identifier un composant en se basant sur une directive de la carte obtenue préalablement en décidant pour chaque directive si elle est réutilisable en dehors du contexte de sa méthode d'origine.

- *Définir un composant de méthode* aide l'ingénieur de méthodes à définir un composant de méthode à partir d'une directive déclarée réutilisable précédemment en lui associant les parties de produits nécessaires à sa réalisation ainsi qu'un descripteur décrivant le contexte de sa réutilisation.

La carte du processus de ré-ingénierie présentée à la Figure 45 décrit un processus itératif qui consiste à définir les sections de la carte de méthode concernée en s'appuyant sur la structure du modèle de produit de la méthode et / ou sur le processus proposé par la méthode puis à définir les différents types de directives associées. Ensuite, il consiste à identifier parmi ces directives celles qui sont réutilisables indépendamment de la méthode elle-même et de les définir en tant que composants de méthode. Le processus permet également de décomposer certaines directives (celles déjà déclarées comme étant de type composant) ou de les agréger afin d'obtenir d'autres directives réutilisables, par conséquent, d'autres composants de méthode de niveau de granularité plus fin ou plus gros que les composants de départ.

La partie de la carte concernant la reconstruction du modèle de processus d'une méthode sous forme d'une directive stratégique est inspirée de la méta-démarche proposée par A. Benjamen dans sa thèse intitulée *"Une Approche Multi-démarches pour la modélisation des démarches méthodologiques"* [Benjamen 99].

L'analyse structurelle du modèle de produit de la méthode, combinée à la réutilisation d'un ensemble de verbes d'intentions de base utilisée par la *stratégie structurelle*, permet la définition des sections de la carte de méthode qui n'a pas du modèle de processus initialement ou si celui-ci se limite à une description informelle du produit à construire. Dans le cas contraire, si la méthode propose un modèle de processus plus sophistiqué, composé des étapes et des fonctions à réaliser, et suggérant différents moyens et/ou différentes manières pour les effectuer afin d'obtenir le produit, le processus de ré-ingénierie propose une autre stratégie, *stratégie fonctionnelle*, qui permet d'identifier les sections de la nouvelle carte en se basant sur ces étapes, ces fonctions, ces moyens et ces manières.

Ensuite, l'ingénieur de méthodes peut soit définir les directives associées aux sections, soit définir de nouvelles sections à partir des sections existantes. La définition des directives associées aux sections consiste à décrire:

- la directive de réalisation d'intention (DRI) associée à chaque section de la carte,
- la directive de sélection d'intention (DSI) associée à un ensemble de sections dont les intentions sources sont identiques et les intentions cibles différentes,
- la directive de sélection de stratégie (DSS) associée à un ensemble de sections dont les intentions sources et les intentions cibles sont identiques mais les stratégies différentes.

Afin de définir ces directives, deux stratégies sont proposées, *stratégie d'utilisation de formulaire* et *stratégie guidée*. La *stratégie d'utilisation de formulaire* consiste à utiliser des formulaires selon le type de la directive que l'ingénieur de méthode doit définir. La *stratégie guidée* propose d'aider l'ingénieur à trouver le type de directive à utiliser pour construire la directive concernée.

La définition de nouvelles sections à partir de sections existantes consiste :

- soit à décomposer les sections existantes,
- soit à agréger les sections existantes,
- soit à trouver des alternatives aux sections existantes,
- soit à progresser à partir de sections existantes.

Ces quatre différentes manières de définition d'une section sont respectivement représentées par quatre stratégies : *stratégie de découverte par regroupement*, *stratégie de découverte par décomposition*, *stratégie de découverte d'alternative*, *stratégie de découverte par progression*. La *stratégie de découverte par regroupement* consiste à regrouper un ensemble de sections en une seule. A l'inverse la *stratégie de découverte par décomposition* consiste à décomposer une section en plusieurs. La *stratégie de découverte d'alternative* permet de déterminer à partir d'une section, de nouvelles sections alternatives en terme d'intention et de stratégie. Enfin, la *stratégie de découverte par progression* propose de trouver de nouvelles sections permettant de continuer dans la démarche à partir d'une section existante.

De plus, la définition d'une section peut se faire par l'analyse d'une directive, selon la description de la directive, il peut être nécessaire de corriger les sections en conséquence. Par exemple, une directive de réalisation d'intention décrite sous forme de plan peut nécessiter la décomposition en plusieurs sections de la section à laquelle la DRI est associée. De même, si la DRI est décrite sous forme de choix, la section associée peut être décomposée en sections alternatives. Cela est pris en compte dans la carte par la *stratégie de correction* utilisée entre les intentions *Définir une directive* et *Définir une section.*

Le processus de ré-ingénierie est centré sur la définition et la modification des sections de la carte de la méthode que l'ingénieur de méthodes cherche à définir. Les modifications sur les sections peuvent nécessiter des modifications des directives associées. La *stratégie de modification* proposée entre les intentions *Définir une section* et *Définir une directive* permet de guider l'ingénieur dans ces modifications en prenant en compte les modifications appliquées aux sections.

L'identification des composants à partir de la directive stratégique de la méthode est soutenue par trois stratégies : la *stratégie de découverte par section*, la *stratégie de découverte par sections parallèles*, la *stratégie de découverte par enchaînement de sections*.

Etant donné qu'une méthode est vue par notre méta-modèle de méthodes *(Chapitre 3)* comme un composant de type agrégat et suivant la définition du composant de méthode proposée par ce méta-modèle, chaque section de la carte de la méthode devient un composant. Dans le cas contraire, la méthode est un composant atomique et aucune de ses sections n'est un composant. La *stratégie de découverte par section* est basée sur ce raisonnement. La *stratégie de découverte par sections parallèles* est basée sur la recherche de directives qui appartiennent aux sections parallèles, c'est-à-dire, celles qui ont la même intention source et la même intention cible mais des stratégies différentes, et leur agrégation afin d'obtenir un composant agrégat. De la même manière la *stratégie de découverte*

par enchaînement de sections propose de sélectionner des directives associées aux sections qui s'enchaînent et de construire ainsi un composant agrégat à base de ces directives.

Toute directive désignée réutilisable est définie en tant que composant de méthode. Le modèle de processus de ré-ingénierie propose deux manières différentes pour définir un composant de méthode :

- avec la *stratégie d'utilisation de formulaire* et
- avec la *stratégie guidée*.

Finalement, le processus de ré-ingénierie peut être arrêté en choisissant la stratégie de complétude qui permet de vérifier si toutes les directives de la carte de la méthode ont été définies, si toutes les combinaisons de directives ont été examinées pour découvrir des composants et si tous les composants identifiés ont été décrits.

2.2 Directives associées à la carte du processus de ré-ingénierie

Les directives associées à la carte du processus de ré-ingénierie sont exprimées de la même manière que les directives associées à n'importe quelle carte décrivant le modèle de processus d'une méthode.

Nous proposons d'abord les signatures des directives associées à la carte du processus de ré-ingénierie, puis nous les décrivons en détail. La Figure 46 comporte la liste complète des signatures de ces directives.

Directives de Réalisation d'Intention :	
<(DIM[2] avec état (DIM) = non traité), Définir une section avec la stratégie structurelle>	DRI1
<(DSM[3] avec état (DSM) = non traité), Définir une section avec la stratégie fonctionnelle>	DRI2
<(Section, avec état (Section) = définie), Définir une section avec la stratégie de découverte par regroupement>	DRI3
<(Section, avec état (Section) = définie), Définir une section avec la stratégie de découverte par décomposition>	DRI4
<(Section, avec état (Section) = définie), Définir une section avec la stratégie de découverte par progression>	DRI5
<(Section, avec état (Section) = définie), Définir une section avec la stratégie de découverte d'alternative>	DRI6
<(Section, avec état (Section) = définie), Définir une directive avec la stratégie d'utilisation de formulaire>	DRI7
<(Section, avec état (Section) = définie), Définir une directive avec la stratégie guidée>	DRI8
<(Section, avec état (Section) = définie), Définir une directive avec la stratégie de modification>	DRI9
<(Directive, avec état (Directive) = définie), Définir une section avec la stratégie de correction>	DRI10
<(DRI, avec état (DRI) = définie), Identifier un composant avec la stratégie de découverte par section>	DRI11
<({DRI, avec état (DRI) = définie}), Identifier un composant avec la stratégie de découverte par sections parallèles>	DRI12
<({DRI, avec état (DRI) = définie}), Identifier un composant avec la stratégie de découverte par enchaînement de sections>	DRI13
<(Composant, avec état (Composant) = identifié), Identifier un composant avec la stratégie de découverte par décomposition>	DRI14
<({Composant, avec état (Composant) = identifié}), Identifier un composant avec la stratégie de découverte par agrégation>	DRI15
<(Composant, avec état (Composant) = identifié), Définir un composant avec la stratégie d'utilisation de formulaire>	DRI16
<(Composant, avec état (Composant) = identifié), Définir un composant avec la stratégie guidée>	DRI17
<(Composant, avec état (Composant) = définie), Définir un composant avec la stratégie de complétude>	DRI18
<(Composant, avec état (Composant) = identifié), Arrêter avec la stratégie de vérification>	DRI19
Directives de Sélection d'Intention :	
<(Méthode avec état (Méthode) = modélisation initiale), Progresser de Démarrer>	DSI1
<(Section, avec état (Section) = définie), Progresser de Définir une section>	DSI2

[2] DIM – Description informelle de la méthode

[3] DSM – Description structurée de la méthode

<(Directive, avec état (Directive) = définie), Progresser de Définir une directive>	DSI3
<(Composant, avec état (Composant) = identifié), Progresser de Identifier un composant>	DSI4
<(Composant, avec état (Composant) = identifié), Progresser de Définir un composant>	DSI5
Directives de Sélection de Stratégie :	
<(Méthode avec état (Méthode) = modélisation initiale), Progresser vers Définir une section>	DSS1
<(Section, avec état (Section) = définie), Progresser vers Définir une section>	DSS2
<(Section, avec état (Section) = définie), Progresser vers Définir une directive>	DSS3
<(Directive, avec état (Directive) = définie), Progresser vers Identifier un composant>	DSS4
<(Composant, avec état (Composant) = identifié), Progresser vers Définir un composant>	DSS5

Figure 46 : Directives associées à la carte de processus de ré-ingénierie

Nous décrivons dans la suite les directives non triviales.

2.2.1 Directives de Réalisation d'Intention du processus de ré-ingénierie

Chaque section de la carte du processus de ré-ingénierie (Figure 45) a une DRI associée aidant l'ingénieur de méthodes à réaliser l'intention source suivant la stratégie proposée par la section. Nous présentons en détail ces directives par la suite.

2.2.1.1 DRI1 : <(DIM avec état (DIM) = non traité), Définir une section avec la stratégie structurelle>

La stratégie structurelle de définition des sections s'applique dans le cas où la construction du produit de la méthode est décrite de manière informelle. La DRI1 permet à l'ingénieur de méthodes de définir des sections de la carte à partir de la description informelle de la méthode (DIM) y compris celle de son modèle de produit faite de manière plus ou moins structurée. La définition des sections de la carte de méthode est centrée sur la structure du modèle de produit de la méthode. Les éléments de produit permettent de définir les intentions, les liens de précédence entre intentions et ainsi les sections de la carte de méthode.

La Figure 47 illustre la DRI1 qui est une directive tactique de type plan. Ce plan propose tout d'abord d'identifier la signature de la carte de méthode (1), ensuite d'identifier les intentions (2), d'identifier les stratégies pour satisfaire ces intentions (3) et finalement de définir les liens de précédence entre les intentions (4).

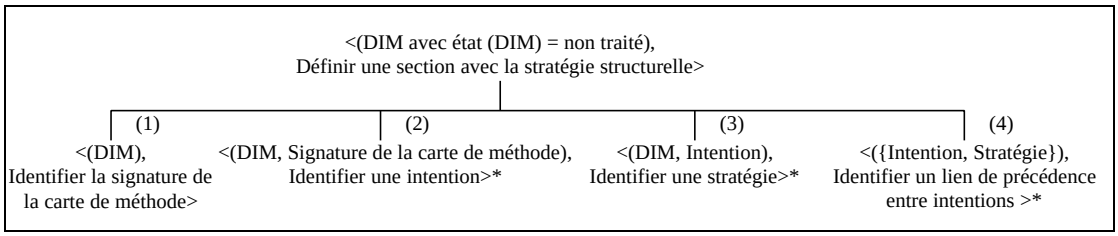


Figure 47: DRI1 : <(DIM avec état (DIM) = non traité), Définir une section avec la stratégie structurelle >

(1) La définition des sections de la carte passe nécessairement par la définition de la signature de la carte. Ceci est réalisé par la directive *<(DIM), Identifier la signature de la carte de méthode>* qui

aide à la définition de la situation et de l'intention de la carte. Cette directive, présentée à la Figure 48, est un plan permettant de définir la situation par instanciation du méta-modèle de méthodes et du méta-modèle de produit (1.1) et de définir l'intention de la carte de méthode (1.2) en utilisant la structure linguistique (verbe, paramètres) proposée par N.Prat [Prat 97], [Prat 99].

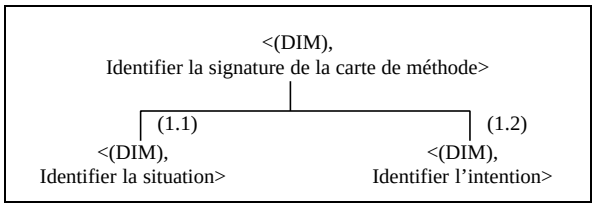


Figure 48: <(DIM), Identifier la signature de la carte de méthode>

(2) La définition de l'intention de la carte réutilise un ensemble de verbes d'intentions standards. L'ingénieur de méthodes peut ainsi définir l'intention en réutilisant un de ces verbes et en déterminant un paramètre à ce verbe par instanciation du méta-modèle de produit.

A partir de la signature de la carte de méthode et de la description informelle de la méthode, l'ingénieur de méthodes peut alors identifier l'ensemble des intentions nécessaires pour la réalisation de l'intention de la méthode dans la situation donnée de la signature. La directive *<(DIM, Signature de la carte de méthode), Identifier une intention>*, guide l'ingénieur de méthodes dans cette identification en proposant d'utiliser la structure du produit de la méthode et une liste de verbes d'intention standards. Cette directive est illustrée par la Figure 49. C'est un plan proposant d'identifier d'abord une partie de produit qui est essentielle dans la structure du produit à construire (2.1), ensuite d'identifier les manipulations nécessaires à effectuer sur cette partie de produit en termes de verbes d'intention (2.2), puis de combiner la partie de produit et les verbes sélectionnés afin d'obtenir les intentions de la carte de méthode (2.3) et finalement sélectionner la ou les intentions qui semblent pertinentes dans la carte de la méthode concernée.

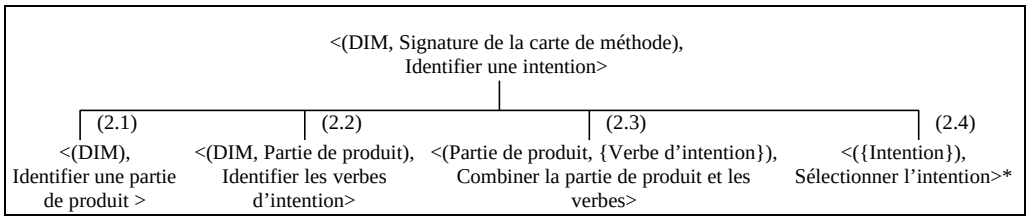


Figure 49: <(DIM, Signature de la carte de méthode), Identifier une intention>

L'identification des intentions à partir de la structure de produit de la méthode est basée sur la définition des parties de produit (2.1) qui sont fondamentales dans cette structure. On suggère à l'ingénieur de méthodes de commencer par la définition de ces parties de produit par instanciation du méta-modèle de produit. Pour cela, il doit en particulier définir les éléments de produit dits *constructionnels*. Les éléments de produit qui participent à la structuration du produit sont dits *constructionnels* et ceux qui ne contribuent qu'à la définition d'un élément de produit sont dits *définitionnels*. Les éléments de produit constructionnels sont les éléments qui apparaissent dans les intentions d'une carte. Par exemple dans le cadre de CREWS-*L'Ecritoire*, les buts et les scénarios

sont des éléments de produit dit *constructionnels* car ils forment la structure du produit de la méthode.

L'identification des verbes d'intention (2.2) se fait à partir d'une liste de verbes prédéfinis. Cette liste de verbes [Plihon 98], issue du projet CREWS[4], a été adaptée à la problématique de l'ingénierie des méthodes. Chaque verbe comporte sa définition et un ensemble de synonymes. Par exemple, dans la carte de la méthode CREWS-*L'Ecritoire*, nous n'avons utilisé que les verbes, *Découvrir, Ecrire*, et *Conceptualiser*. Comme le montre Figure 50, le verbe *Découvrir* est un synonyme du verbe *Elucider*, tandis qu'*Ecrire* est synonyme de *Documenter*. La liste complète des verbes est donnée en annexe.

Elicit : The process of systematically obtaining from people new facts (scenarios, requirements) about the domain / business processes / the system under consideration.

Synonyms: Acquire, Discover, Capture

Document : As opposed to 'to conceptualise', write down the product of activities such as analyse, compare, change, etc.

Synonyms: Describe, Specify, Write

Conceptualise : The process of systematically abstracting (existing or envisaged) real-world phenomena into models which highlight the essential aspects and hide the unimportant details (relative to the viewpoint taken).

Synonyms: Model, Abstract

Figure 50 : Extrait du glossaire de verbes prédéfinis

Une fois qu'un ou plusieurs verbes sont identifiés, l'ingénieur de méthodes peut combiner la partie de produit et les verbes d'intention précédemment définis (2.3 à la Figure 49). Il obtient ainsi un ensemble d'intentions candidates pour la carte de la méthode concernée. Finalement, il est guidé dans la sélection des intentions parmi les intentions obtenues (2.4 à la Figure 49). Par exemple, pour la méthode CREWS-*L'Ecritoire* il doit décider laquelle des intentions sélectionner parmi les suivantes : *Elucider un but*, *Acquérir un but* ou bien *Découvrir un but*.

(3) A partir de l'ensemble des intentions produites par l'exécution répétitive de la directive *<(DIM, Signature de la carte de méthode), Identifier une intention>* et la description informelle de la méthode la directive *<(DIM, {Intention}), Identifier une stratégie>* illustrée à la Figure 51 permet (grâce au paramètre *manière* de l'intention) de définir les stratégies de la carte. Cette directive est un plan proposant pour chaque intention définie préalablement d'identifier toutes les manières possibles pour réaliser cette intention (3.1 à la Figure 51) et ensuite identifier une stratégie à partir d'une ou plusieurs manières (3.2).

[4] CREWS (ESPRIT N° 21.903) Cooperative Requirements Engineering With Scenarios

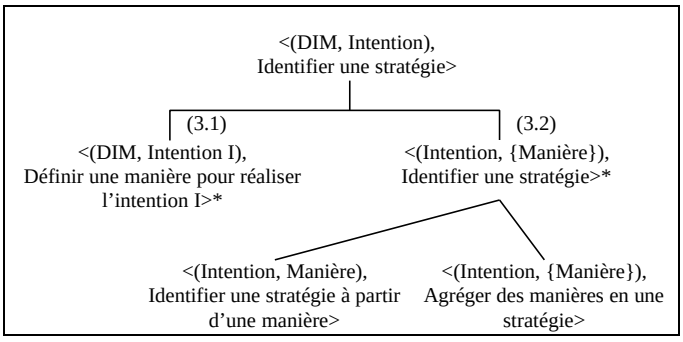


Figure 51 : <(DIM, Intention), Identifier une stratégie>

Les manières pour réaliser une intention peuvent être identifiées à partir de la description informelle de la méthode. S'il existe plusieurs manières pour satisfaire une intention, l'ingénieur de méthodes doit décider si ce sont des stratégies différentes ou seulement des tactiques d'une même stratégie. L'identification d'une stratégie peut se faire par agrégation de manières, ou par sélection d'une manière qui devient alors la stratégie. Pour distinguer ces deux cas il faut vérifier si le produit source et le produit cible sont les mêmes en appliquant des manières différentes. Si c'est le cas, les manières ne sont que des tactiques dans une même stratégie et l'ingénieur de méthodes doit appliquer le processus d'agrégation des manières afin d'obtenir une stratégie. Dans le cas contraire, il définit autant de stratégies qu'il a découvert de manières. Par exemple, CREWS-*L'Ecritoire* propose plusieurs manières pour *"Ecrire un scénario"* comme *"avec les directives de style"*, *"avec les directives de contenu*", *"en appliquant une structure prédéfinie"*. Les deux premières manières utilisent le même produit source, le *"but"*, et produisent des *"scénarios"* qui ont la même structure, tandis qu'en appliquant la troisième on obtient des *"scénarios"* ayant une autre structure. Par conséquent, l'ingénieur de méthodes couple les deux premières manières et obtient une stratégie nommée "prose libre" tandis que la troisième manière devient une stratégie à part entière – *"stratégie de la structure prédéfinie"*.

(4) Enfin, la définition des sections de la carte de méthode passe par l'identification des liens de précédence entre intentions. La directive *<{Intention, Stratégie}, Identifier les liens de précédence entre intentions>* (Figure 52) permet l'identification des sections par définition de la situation d'application de l'intention. Cette directive est un plan qui exécute en séquence:

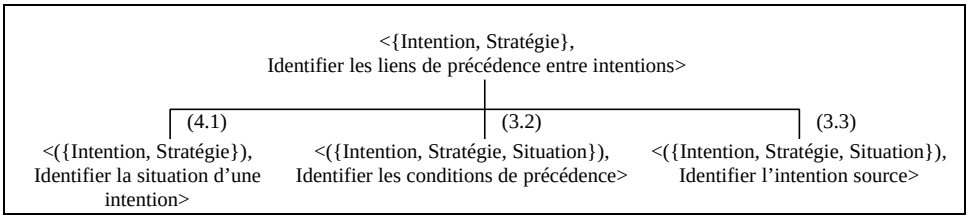


Figure 52: <({Intention, Stratégie}), Identifier les liens de précédences entre intentions>

Cette directive suggère d'identifier d'abord la situation d'une intention, c'est-à-dire la ou les parties de produit nécessaires à l'exécution de l'intention à partir des éléments de produit de la méthode (4.1). Par exemple, l'intention *"Ecrire un scénario"* avec la stratégie *"de prose libre"* dans CREWS-*L'Ecritoire* a pour situation le concept *"but"* dont la réalisation doit être décrite par

un scénario. Ensuite, l'application de la directive exécutable *<{Intention, Stratégie, Situation}, Identifier les conditions de précédence d'une intention>* permet de définir à l'aide du méta-modèle de produit, les conditions nécessaires sur les éléments de produit définis précédemment. Pour notre exemple, il est nécessaire d'avoir un but défini suivant le modèle de but et non décrit par un scénario, la situation se note alors *(But avec état(But) = structuré et non décrit)*. Une fois les situations de précédence complètement définies, l'ingénieur de méthodes est invité à lier les intentions les unes aux autres en vérifiant que l'exécution d'une intention source produit les éléments de la situation d'une intention cible. Ainsi, les sections de la carte de méthode sont définies.

2.2.1.2 DRI2 : <(DSM avec état (DSM) = non traité), Définir une section avec la stratégie fonctionnelle>

La définition des sections avec la stratégie fonctionnelle s'applique dans le cas où le modèle de processus de la méthode est décrit de manière plus ou moins structurée, par des étapes et des opérations à réaliser afin d'obtenir le produit proposé par la méthode. La DRI2 permet à l'ingénieur de méthodes de définir des sections de la carte à partir du modèle de ces étapes et de ces opérations, c'est-à-dire à partir de la description structurée de la méthode *(DSM)*. La définition des sections de la carte de méthode est centrée sur la structure du modèle de processus de la méthode. Les étapes à effectuer dans la démarche de la méthode permettent de définir les intentions, les liens de précédence entre intentions et ainsi les sections de la carte de méthode.

Comme le montre la Figure 53 illustrant la DRI2, cette directive a la même structure que la DRI1 mais le produit de départ n'est pas le même. Alors que la DRI1 travaille sur la description informelle de la méthode, la DRI2 est basée sur l'analyse de la description plus structurée qui fait apparaître les étapes de procédé, les fonctions à réaliser et/ou les opérations à exécuter pour construire le produit proposé par la méthode. Comme la DRI1, la DRI2 propose de démarrer le processus de définition des sections de la carte de méthode par l'identification de sa signature ce qui est identique à DRI1. Ce sont l'identification des intentions et l'identification des stratégies qui sont différentes car elles sont basées sur le modèle de processus et non sur le modèle de produit de la méthode. La définition des liens de précédence entre les intentions reste la même que celle proposée par la DRI1. Nous allons développer alors les sous-directives (2) et (3) de la DRI1.

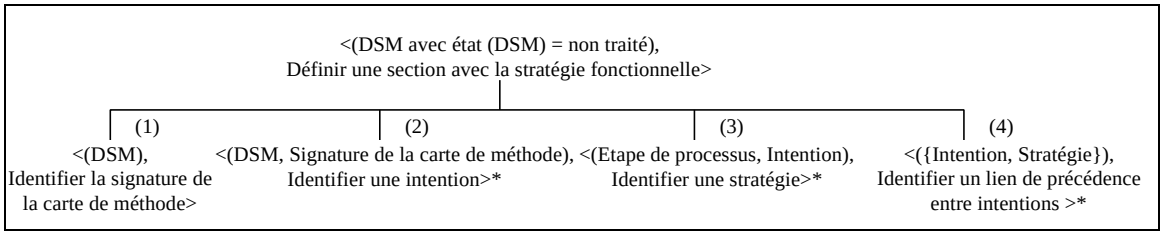


Figure 53 : DRI2 : <(DSM avec état (DSM) = non traité), Définir une section avec la stratégie fonctionnelle>

(2) A partir de la signature de la carte de méthode et de la description par étapes ou par fonctions du modèle de processus de la méthode, l'ingénieur de méthodes peut identifier les intentions à

réaliser afin d'obtenir le produit de la méthode. La directive <(DSM, Signature de la carte de méthode), Identifier une intention> illustrée à la Figure 54 guide l'ingénieur de méthodes dans cette identification en lui proposant de commencer par identifier les principales étapes (ou fonctions) dans la description structurée du modèle de processus de la méthode (2.1) avec l'objectif de transformer chaque étape en une intention dans la carte de la méthode. Ensuite il est invité à identifier le produit qui est construit ou modifié durant cette étape (2.2), à désigner l'étape par un ou plusieurs verbes d'intention (2.3), à construire l'intention en combinant la partie de produit avec les verbes sélectionnés (2.4) et à sélectionner une ou plusieurs intentions qui lui semblent les plus pertinentes pour représenter l'étape de processus concerné.

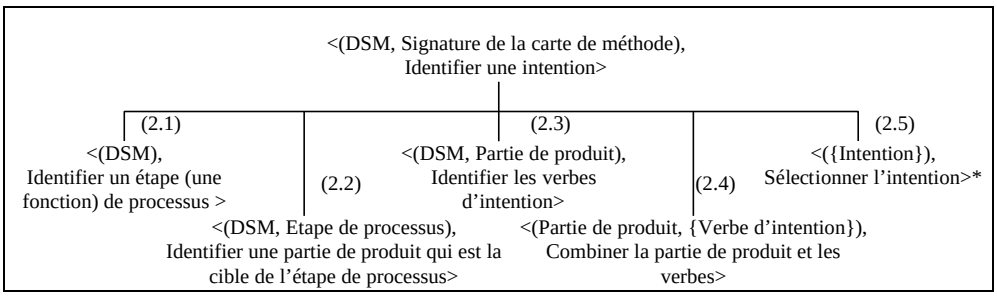


Figure 54: <(DSM, Signature de la carte de méthode), Identifier une intention>

L'identification des intentions à partir du processus structuré consiste à identifier les principales étapes de ce processus en commençant par le niveau d'abstraction le plus haut possible en sachant que chaque étape doit construire ou modifier une des parties de produit. Si à un niveau donné les étapes proposent déjà plusieurs manières pour manipuler les parties de produit concernées, on peut s'arrêter à ce niveau d'abstraction. Sinon, il est conseillé de descendre d'un niveau, c'est-à-dire de décomposer chaque étape en sous étapes, surtout si les étapes manipulent des parties de produit complexes qui peuvent être décomposées et traitées séparément.

Une fois que l'ingénieur de méthodes a identifié les étapes de processus, il peut identifier quelle est la partie de produit cible de chaque étape en se basant sur le modèle de produit de la méthode. Comme dans la DRI1, ces parties de produit doivent être de type constructionnel.

L'identification des verbes d'intention se fait de la même manière que dans la DRI1 mais en respectant la sémantique de l'étape du processus concernée. La suite, c'est-à-dire la construction des intentions par combinaison de la partie de produit et des verbes sélectionnés se fait de la même manière que dans la DRI1. Normalement, l'ingénieur de méthodes ne doit sélectionner qu'une section par étape de processus, sinon ça reviendrait à la décomposition de l'étape en sous-étapes.

(3) L'identification des stratégies est également basée sur l'analyse du modèle de processus de la méthode. Plus exactement, l'ingénieur de méthodes est invité à analyser chaque étape du processus et à découvrir s'il propose plusieurs manières pour satisfaire l'intention qui était identifiée dans cette étape. La directive *<(Etape de processus, Intention), Identifier une stratégie>* est illustrée par la Figure 55.

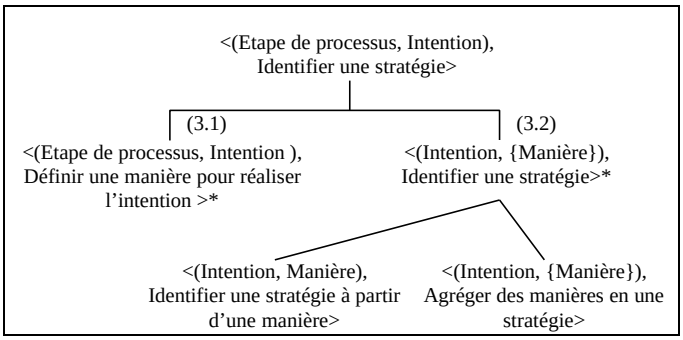


Figure 55 : <(Etape de processus, Intention), Identifier une stratégie>

Chaque étape de processus peut proposer une ou plusieurs manières pour réaliser l'intention de l'étape. En suivant le même raisonnement que celui proposé par la DRI1 l'ingénieur de méthodes doit vérifier si chaque manière identifiée correspond à une stratégie ou s'il faut composer plusieurs manières différentes pour identifier une stratégie.

2.2.1.3 DRI3 : <(Section, avec état (Section) = définie), Définir une section avec la stratégie de découverte par regroupement>

La DRI3 permet de définir une nouvelle section à partir d'un ensemble de sections. La stratégie de découverte consiste soit à regrouper les intentions des sections soit les stratégies des sections. La DRI3 présentée à la Figure 56 est décrite sous forme d'un choix contenant deux alternatives : définir une nouvelle section en regroupant les stratégies ou définir une nouvelle section en regroupant les intentions.

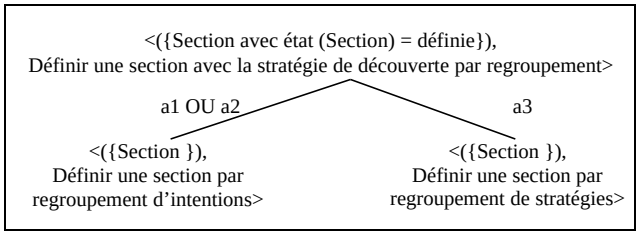


Figure 56: DRI3 : <({Section avec état (Section) = définie}), Définir une section avec la stratégie de découverte par regroupement>

L'alternative *<({Section}), Définir une section par regroupement d'intentions>* permet de regrouper les intentions cibles d'un ensemble de sections si l'ingénieur de méthodes juge que ces intentions sont d'un niveau d'abstraction trop bas par rapport aux intentions des autres sections de la carte de méthode (a1), ou si les sections qui se suivent n'ont pas d'alternative de stratégie et qu'elles représentent une séquence d'intentions à exécuter (a2). Cette alternative est représentée par la Figure 57. Elle permet de sélectionner les sections à regrouper dans l'ensemble des sections, de sélectionner parmi celles-ci les intentions à regrouper, de regrouper ces intentions dans une même intention, puis de regrouper les stratégies impliquées, et enfin de modifier les sections qui avaient comme intention, une intention regroupée. Par exemple,

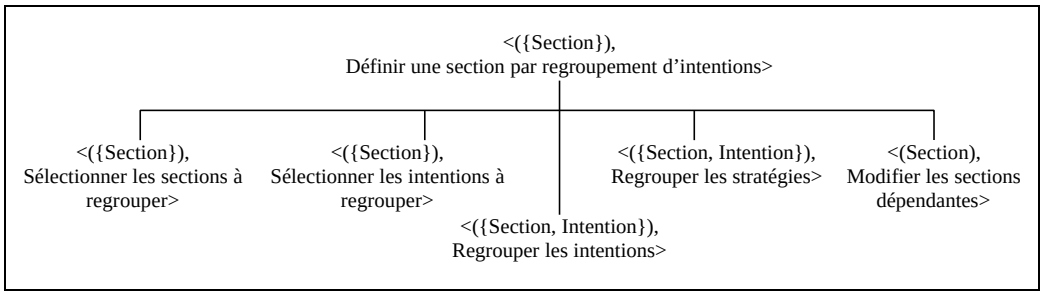


Figure 57: <({Section}), Définir une section par regroupement d'intentions>

Le regroupement d'intentions utilise la formalisation des intentions sous forme de (verbe, paramètre) présentée au Chapitre 3. Ceci permet à l'ingénieur de regrouper des intentions en se basant soit sur le verbe et soit sur les paramètres. Les paramètres de type résultat (Chapitre 3) faisant référence à des éléments de produit peuvent être agrégés afin d'augmenter le niveau de granularité. Il est en particulier possible d'utiliser un élément de produit agrégé contenu dans le modèle de produit. Par exemple, les intentions *Ecrire un scénario normal* et *Ecrire un scénario exceptionnel*, peuvent être regroupées en une intention *Ecrire un scénario* qui représente une abstraction des deux intentions initiales.

L'alternative *<({Section}), Définir une section par regroupement de stratégie*> est une directive exécutable qui consiste à regrouper des stratégies de sections ayant les mêmes intentions sources et cibles mais des stratégies différentes qui peuvent être regroupées (a3). Cette directive consiste à supprimer les deux sections et à en définir une nouvelle ayant un nom de stratégie qui englobe les deux dernières.

2.2.1.4 DRI4 : <(Section, avec état (Section) = définie), Définir une section avec la stratégie de découverte par décomposition>

A l'inverse de la DRI3, la DRI4 permet de décomposer une section en plusieurs sections. Cette décomposition peut porter sur les intentions d'une section ou sur les stratégies d'une section. La DRI4 présentée à la Figure 58 propose donc un choix contenant deux alternatives pour définir une section :

- par décomposition d'intention ou
- par décomposition de stratégie.

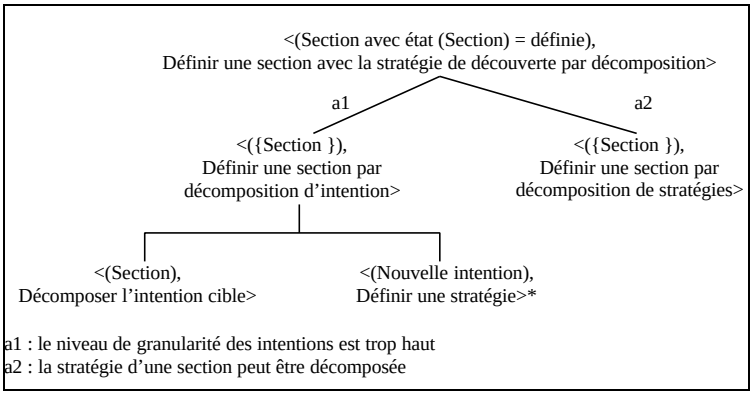


Figure 58: DRI4 : <(Section), Définir une section avec la stratégie de découverte par décomposition>

La première alternative *<(Section), Définir une section par décomposition d'intention>* permet de définir de nouvelles sections en décomposant l'intention cible de la section en plusieurs intentions si l'ingénieur de méthodes juge que l'intention est de trop haut niveau de granularité (a1). Ensuite, il est nécessaire de définir pour chaque couple d'intentions, la stratégie de la nouvelle section.

La décomposition de l'intention utilise la formalisation des intentions (verbe, paramètres). La décomposition des paramètres utilise la décomposition des éléments de produit composés. Les éléments de produit composant peuvent être utilisés comme nouveaux paramètres du verbe. Par exemple, la section *<Démarrer, Construire un fragment de besoins, Stratégie linguistique>* est ainsi décomposée en *<Démarrer, Découvrir un but, Stratégie de modèle de but>* et *<Découvrir un but, Ecrire un scénario, Stratégie de prose libre>* car l'élément de produit composé *Fragment de besoin* est composé d'un but et d'un scénario correspondant. Il faut noter que les verbes d'intention et les stratégies des nouvelles sections ont été renommés pour rendre l'expression plus explicite.

La définition de la stratégie propose deux alternatives, celle décrite ci-dessus à la Figure 51 ou celle de la Figure 55 suivant le cas. Si la méthode concernée est décrite de manière informelle, c'est l'alternative de la Figure 51 qui doit être appliquée. Par contre, si la méthode propose un modèle de processus basé sur des étapes à réaliser, c'est l'alternative de la Figure 55 qui doit être appliquée.

La deuxième alternative *<(Section), Définir une section par décomposition de stratégie>* (Figure 58) est une directive exécutable qui décompose la stratégie d'une section en plusieurs stratégies permettant de prendre en compte ou de générer une situation particulière. Par exemple, la section *<Découvrir un but, Ecrire un scénario, Stratégie linguistique de* CREWS-*L'Ecritire>* peut être décomposée en deux sections *<Découvrir un but, Ecrire un scénario, Stratégie de prose libre>* et *<Découvrir un but, Ecrire un scénario, Stratégie de structure prédéfinie>*

2.2.1.5 DRI5 : <(Section, avec état (Section) = définie), Définir une section avec la stratégie de découverte par progression>

La DRI5 sert à la découverte des sections qui suivent une section donnée, c'est-à-dire qu'elle permet de trouver des sections permettant de progresser à partir de la situation engendrée par l'exécution de la section donnée. Cette directive est présentée à la Figure 59. Elle est décrite sous forme d'un plan contenant trois éléments:

- l'analyse de l'intention cible de la section concernée,
- la définition d'une nouvelle intention utilisant en tant que produit source le produit obtenu en exécutant l'intention considérée et
- la définition de la stratégie permettant de satisfaire la nouvelle intention à partir de l'intention du départ.

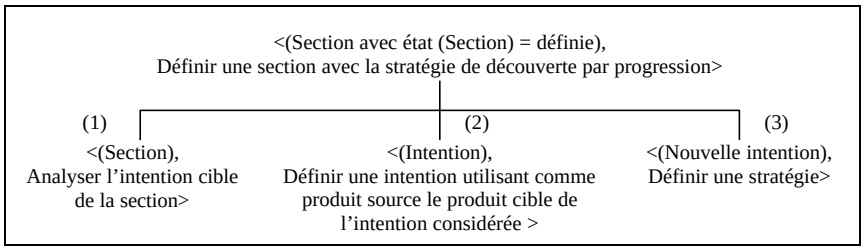


Figure 59: DRI5 : <(Section avec état (Section) = définie), Définir une section avec la stratégie de découverte par progression >

L'analyse de l'intention cible de la section porte sur le paramètre cible de l'intention analysée en terme d'une partie de produit construite ou modifiée par l'intention (1). Cette analyse peut être basée sur la description informelle du produit de la méthode ou sur le modèle de processus structuré en étapes. Dans le premier cas l'ingénieur de méthodes cherche s'il existe un lien entre la partie de produit qui est la cible de l'intention considérée et une autre partie de produit. Si c'est le cas, il peut être utile de définir une nouvelle intention contenant cette partie de produit. Dans le cas contraire, l'ingénieur de méthodes doit chercher une autre étape de processus qui utilise cette partie de produit en tant qu'un produit source et qui la transforme ou crée une autre partie de produit à base de celle-ci. Puis la directive aide à définir la nouvelle intention (2) et une stratégie permettant de réaliser cette intention à partir de l'intention cible de la section considérée (3). La définition de la stratégie est identique à celle proposée par la DRI4

2.2.1.6 DRI6 : <(Section, avec état (Section) = définie), Définir une section avec la stratégie de découverte d'alternative>

La DRI6 aide à la découverte de sections alternatives à une section considérée, elle permet en particulier de trouver des sections ayant soit des stratégies alternatives soit des intentions cibles alternatives. Elle est présentée à la Figure 60 où elle est décrite sous forme d'un choix contenant deux alternatives qui permettent de découvrir des sections alternatives à une section donnée.

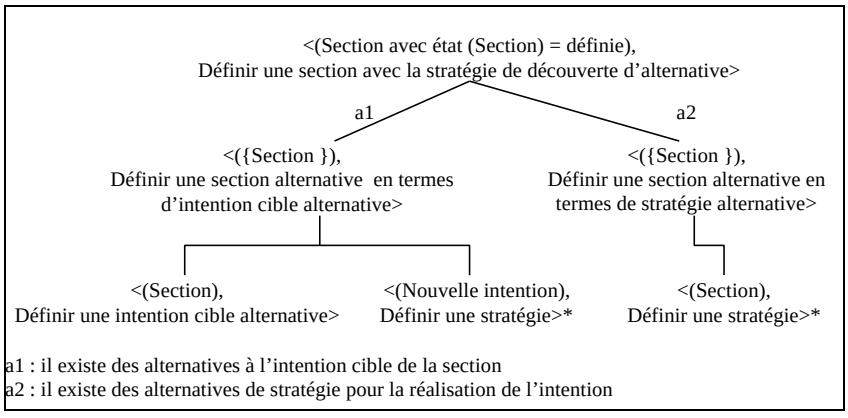


Figure 60: DRI6 <(Section avec état (Section) = définie), Définir une section avec la stratégie de découverte d'alternative>

L'exécution de la première alternative consiste à définir une intention alternative à l'intention cible de la section donnée en utilisant la structure (verbe, cible) de l'intention et en jouant sur ces deux paramètres en fonction du modèle de produit de la méthode. Si la cible du verbe est une partie de produit P1 élément d'une autre partie de produit composé Pc, il est souvent possible d'utiliser une des autres parties de produit P2 élément de la partie de produit composé Pc comme paramètre de l'intention alternative. De même, si le paramètre du verbe est une partie de produit P1 spécialisée d'une partie de produit P donné, il est souvent possible d'utiliser une autre partie de produit P2 spécialisée de la partie de produit P comme paramètre de l'intention alternative. Une fois que l'intention alternative est identifiée, il est nécessaire de définir au moins une stratégie de réalisation de cette intention. Le processus de la définition d'une stratégie est identique à celui utilisé par les DRI4 et DRI5.

La deuxième alternative proposée par la DRI6 (Figure 60) permet, à partir de l'intention cible de la section, de définir des stratégies alternatives pour réaliser cette intention en appliquant le même processus de définition de stratégie que dans les DRI précédentes.

2.2.1.7 DRI7 : <(Section, avec état (Section) = définie), Définir une directive avec la stratégie d'utilisation de formulaire>

La DRI7 permet de définir une directive associée à une ou plusieurs sections en utilisant des formulaires prédéfinis. Un des formulaires est présenté à la Figure 61.

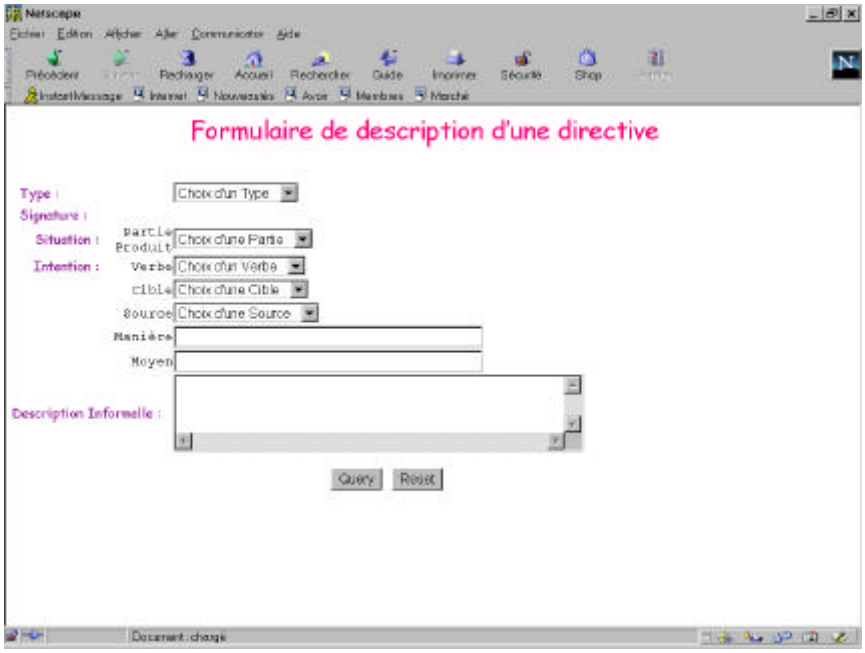


Figure 61: Formulaire de description d'une directive associée à la carte de méthode

Le formulaire permet de décrire les différents éléments caractérisant une directive. De plus, un ensemble de valeurs prédéfinies est proposé à l'ingénieur de méthode pour faciliter sa tâche.

2.2.1.8 DRI8 : <(Section, avec état (Section) = définie), Définir une directive avec la stratégie guidée>

La définition d'une directive avec la stratégie guidée a pour objectif de guider l'ingénieur de méthodes à formaliser la directive quel que soit son type (DRI, DSI, DSS) et quel que soit son type structurel (stratégique, tactique ou simple).

La DRI8 est décrite à la Figure 62. C'est un plan proposant d'identifier d'abord le type de la directive (1) puis de définir sa signature ensuite (2) et enfin de définir son corps (3).

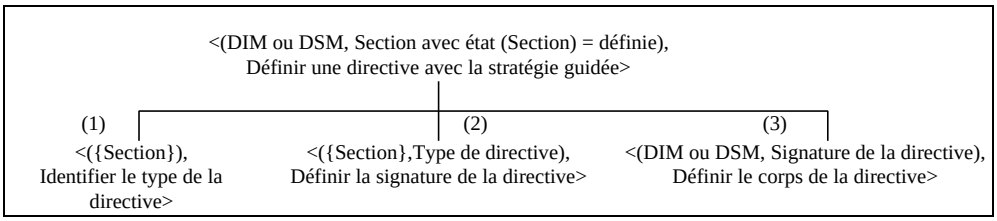


Figure 62: DRI8 : <({Section avec état (Section) = définie}), Définir une directive avec la stratégie guidée>

Identifier le type de la directive consiste à définir une directive de type DRI, DSI ou DSS (Chapitre 3). L'ingénieur de méthodes a donc un choix entre trois alternatives possibles ce qui est illustré à la Figure 63.

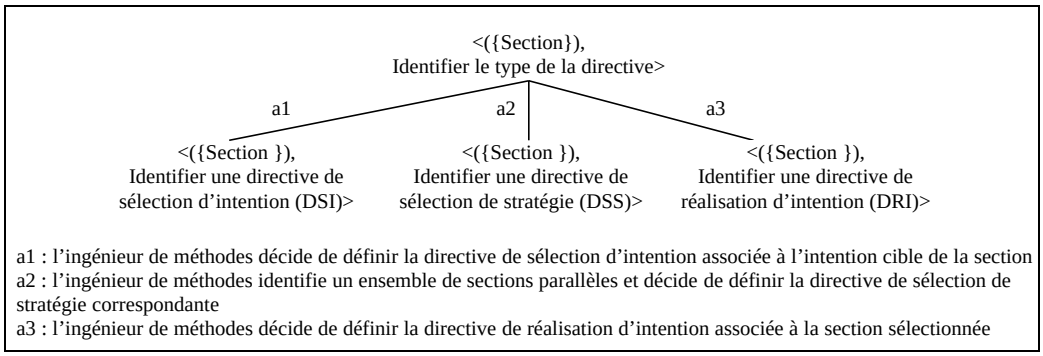


Figure 63: <({Section}), Définir le type de la directive>

Une fois qu'il a défini une section, l'ingénieur de méthodes peut définir la DRI associée à cette section. La directive DSS peut être définie seulement après avoir identifié au moins deux sections parallèles, c'est-à-dire des sections ayant la même intention source et la même intention cible. Finalement, il est conseillé de définir les DSI après avoir défini toutes les sections de la carte sinon l'identification des nouvelles sections entraînerait de nombreuses modifications de ces directives.

Selon le type de la directive définie, l'ingénieur de méthodes doit préciser sa signature. Il est guidé par la directive *<(Type de la directive), Définir la signature de la directive>* décrite à la Figure 64. Puisque toute signature est un couple <situation, intention> (Chapitre 3), cette directive permet de définir la situation, et l'intention de la directive concernée.

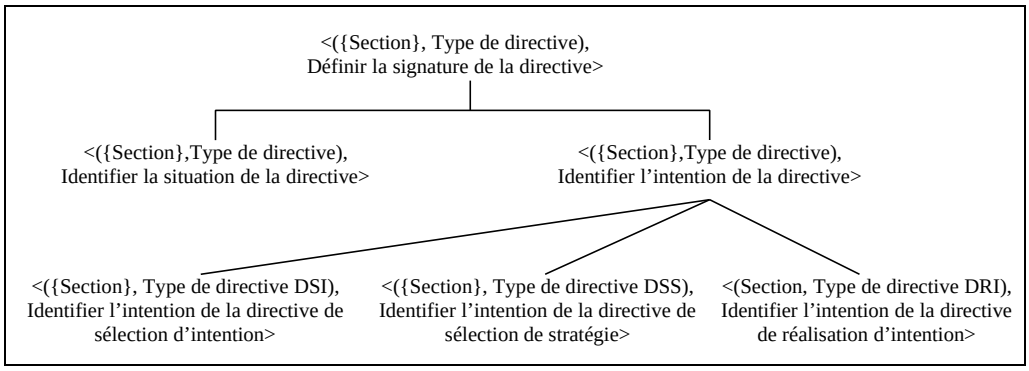


Figure 64: <(Type de la directive), *Définir la signature de la directive*>

La description de la situation et de l'intention reprend les informations contenues dans les sections. La situation est déterminée par les conditions de précédence des intentions. L'intention est définie différemment suivant le type de directive. Si la directive est de type DRI, l'intention est définie en utilisant l'intention cible de la section concernée. Si la directive est de type DSI, l'intention est *"Progresser de I*" où I est une intention source de plusieurs sections ayant des intentions cibles différentes. Finalement, pour la directive de type DSS l'intention est *"Progresser vers I"* où I est une intention cible de plusieurs sections ayant également l'intention source identique.

A partir du type de la directive, de sa signature et de la description de l'étape correspondant dans le modèle de processus initial de la méthode l'ingénieur de méthodes est guidé dans la construction du corps de la directive. Suivant notre méta-modèle de méthodes présenté au Chapitre 3, le corps d'une DRI peut être simple, tactique ou même stratégique tandis que les corps des DSI et DSS sont toujours de type tactique, plus précisément ce sont toujours des directives de type choix. Comme le montre la Figure 65, il est nécessaire d'identifier d'abord le type du corps de la directive aant de démarrer sa construction.

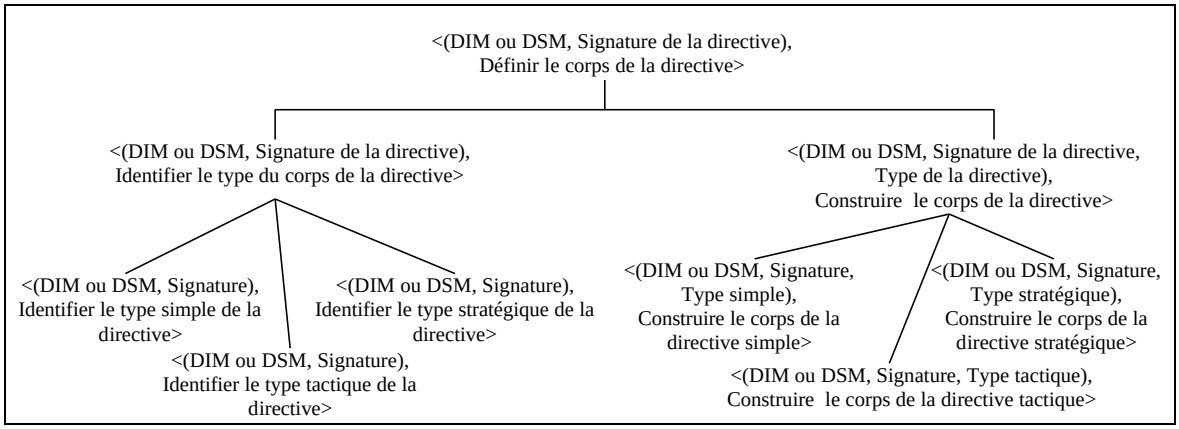


Figure 65: <(DIM ou DSM, Signature de la directive), Définir le corps de la directive>

Si l'identification du type du corps pour les DSI et DSS ne demande aucun effort car on sait que se sont toujours des directives tactiques de type choix, la même identification pour les DRI dépend de la description initiale de la méthode, en particulier de sa richesse et du formalisme utilisé. Si la méthode est présentée d'une manière informelle (DIM), il est parfois très difficile de définir le processus

nécessaire à la réalisation d'une intention par une directive tactique. Dans ce cas, l'ingénieur de méthodes a souvent recours à une directive simple, c'est-à-dire informelle ou au mieux exécutable et comportant des actions à réaliser. Si la méthode propose une description structurée du processus à réaliser (DSM), il peut définir le processus de la réalisation d'intention par une directive tactique. Et c'est seulement dans les cas très rares que l'ingénieur de méthodes arrive à identifier une autre directive stratégique, c'est-à-dire une carte de niveau de granularité plus fine.

La construction du corps d'une directive *simple informelle* consiste à extraire la partie de la description informelle de la méthode concernant la réalisation de l'intention de la directive concernée.

Pour construire une directive *simple exécutable* il est nécessaire d'exprimer le processus de la réalisation d'intention en termes d'une action atomique ou d'un flux d'actions suivant la complexité du processus. Ces actions sont plus faciles à identifier à partir d'une description structurée du processus initial de la méthode qu'à partir d'une description informelle de la méthode. Le méta-modèle de méthodes proposé au Chapitre 3 montre qu'il existe deux types d'actions : *action d'ingénierie* et *action de délégation*. Dans le cas de la description d'une directive de réalisation d'intention (DRI), l'ingénieur de méthodes est toujours guidé vers la description d'une action de transformation car il s'agit de la construction ou de la modification d'une partie de produit.

La construction du corps d'une directive *tactique plan* consiste à décomposer l'intention de la directive en sous-intentions complémentaires de granularité plus fine dont l'ensemble constituerait un plan d'exécution. Pour cela il est nécessaire d'examiner l'étape du modèle de processus de la méthode ou la description informelle de la méthode concernant la réalisation de cette l'intention et de le (la) décomposer en un plan d'opérations à réaliser où chaque opération devient une sous-directive de niveau de granularité plus fine. Chaque sous-directive peut à son tour être une directive simple ou tactique. Le méta-modèle de méthode (Chapitre 3) n'interdit pas l'existence d'une directive stratégique. Mais à ce niveau de granularité il est difficile de raisonner en termes de stratégies. La définition d'une directive tactique plan est illustrée à la Figure 66.

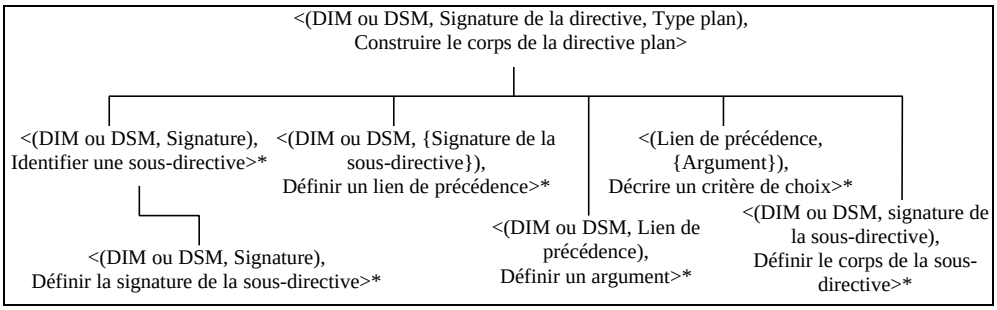


Figure 66: <(DIM ou DSM, Signature de la directive, Type plan), Construire le corps de la directive plan>

Comme le montre la Figure 66, l'ingénieur de méthodes est guidé dans l'identification des sous-directives composant le plan en définissant leurs signatures, la description des liens de précédence entre ces sous-directives, la définition des arguments, la description des critères de choix associés aux liens de précédence, et finalement la définition du corps de chaque sous-directive ce qui est un appel récursif à la démarche proposée par la Figure 65.

La construction du corps d'une directive tactique choix consiste à affiner l'intention de la directive concernée en sous-directives qui représentent des manières alternatives pour réaliser l'intention du départ. L'ingénieur de méthodes examine la description de la méthode concernant la réalisation de l'intention donnée afin d'identifier différentes manières proposées par la méthode pour satisfaire cette intention. Chaque manière devient une sous-directive de la directive de départ qui est une alternative possible de la réalisation de l'intention initiale. Le corps de la directive choix réunit toutes ces alternatives et propose des arguments en faveur de chacune. La construction d'une directive choix est illustrée à la Figure 67.

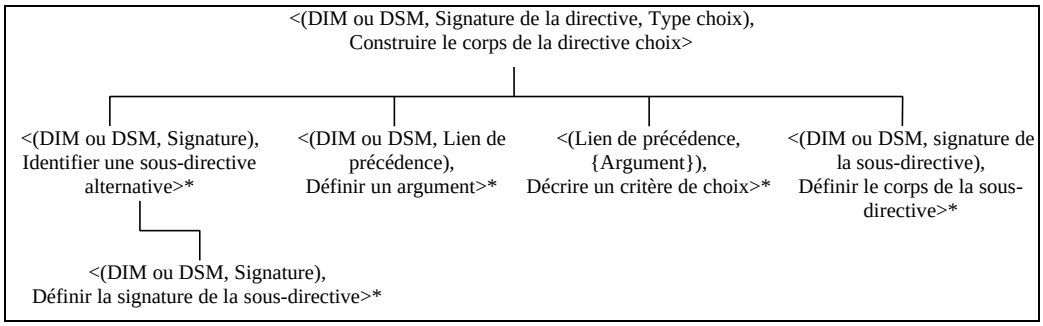


Figure 67: <(DIM ou DSM, Signature de la directive, Type choix), Construire le corps de la directive choix>

La Figure 67 montre comment l'ingénieur de méthodes est guidé pour identifier les alternatives de la réalisation de l'intention du départ en terme de sous-directives, pour définir ensuite des arguments et des critères de choix pour chaque alternative et finalement pour définir le corps de chaque sous directive ce qui est un appel récursif à la démarche proposée par la Figure 65 comme dans le cas d'une directive plan.

La définition des directives plan et choix est récursive car toutes leurs sous-directives peuvent à leur tour être des directives de n'importe quel type. En affinant ou en décomposant progressivement les directives on obtient une hiérarchie de type arbre qui s'arrête quand toutes ses feuilles deviennent des directives simples (informelles ou exécutables).

En ce qui concerne les directives DSI et DSS, ce sont toujours des directives de type choix dont toutes les alternatives sont des directives exécutables décrites chacune par une action de délégation car il s'agit de décrire la progression dans la carte, c'est à dire de sélectionner d'autres directives.

Finalement, la construction du corps d'une directive stratégique n'est qu'une itération de la définition des sections et des directives proposées par le processus de ré-ingénierie à la Figure 45.

2.2.1.9 DRI9 : <(Section, avec état (Section) = définie), Définir une directive avec la stratégie de modification>

La modification des sections peut nécessiter la modification des directives associées déjà définies. La DRI9 aide l'ingénieur de méthodes à modifier les directives en fonction de la modification effectuée sur la section traitée. La modification concerne la DRI de la section traitée, et peut concerner la DSS et

la DSI associées à la section si elles existent. La DRI9 présentée à la Figure 68 est décrite sous forme d'un plan contenant trois sous-directives, relatives à la modification des trois types de directives (DRI, DSI, DSS).

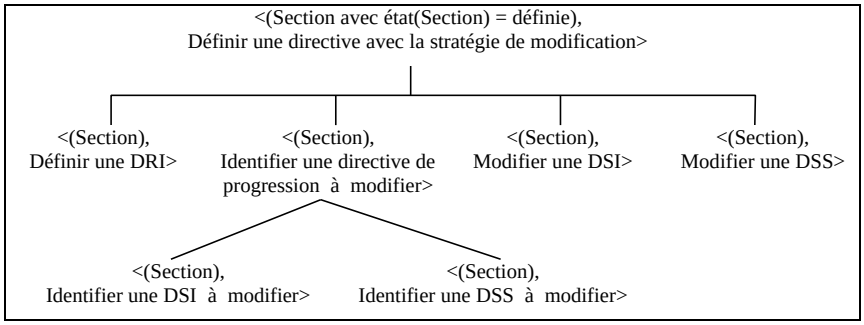


Figure 68: DRI9 <(Section avec état (Section) = définie), Définir une directive avec la stratégie de modification>

La section peut provenir soit d'un regroupement de sections existantes, soit de la décomposition d'une section existante. Dans le premier cas les DRI des sections existantes sont regroupées dans une nouvelle DRI associée à la section traitée. Dans le second, la DRI de la section existante est décomposée pour définir la DRI de la section traitée. Ceci est décrit par un choix à la Figure 69.

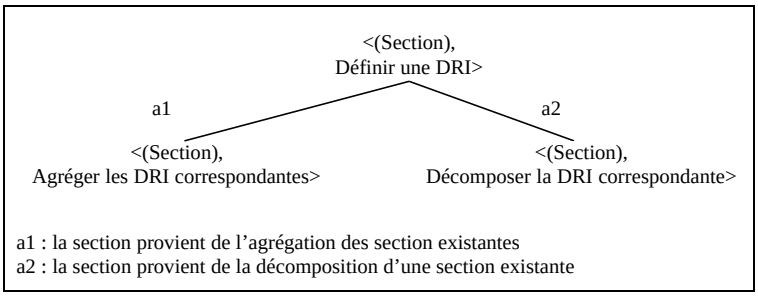


Figure 69: <(Section), Définir une DRI>

La modification des DSI et des DSS existantes consiste, quelle que soit la provenance de la section traitée, à supprimer la ou les alternatives de stratégie ou d'intention décomposées ou regroupées puis à rajouter l'alternative de stratégie ou d'intention définie dans la section traitée dans le contexte choix de la DSI et/ou de la DSS. Ces modifications sont respectivement réalisées suivant les plans représentés à la Figure 70 et à la Figure 71.

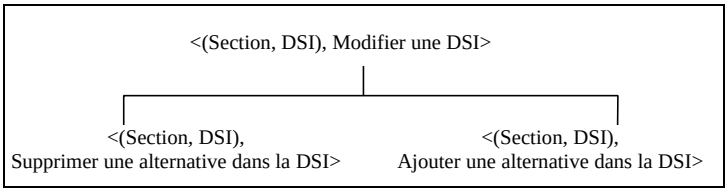


Figure 70: <(Section, DSI), Modifier une DSI>

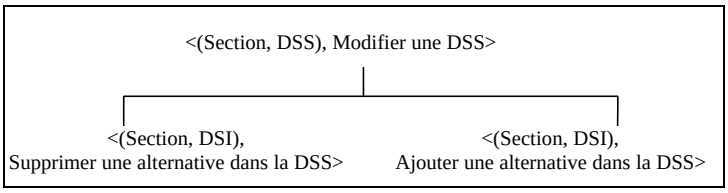


Figure 71: <(Section, DSS), Modifier une DSS>

2.2.1.10 DRI10 : <(Directive, avec état (Directive) = définie), Définir une section avec la stratégie de correction>

La DRI10 aide l'ingénieur de méthodes à corriger les sections définies préalablement. La définition d'une directive de réalisation d'intention associée à une section peut mettre en évidence le fait que la section ne soit pas correcte et qu'elle doive être décomposée en nouvelles sections.

Cette directive, décrite à la Figure 72, est représentée sous forme d'un plan proposant à l'ingénieur de méthodes de supprimer la section pour laquelle la création de sa DRI a mis en évidence la correction à apporter, de définir les nouvelles sections à partir de la description de cette DRI et finalement, de supprimer cette DRI.

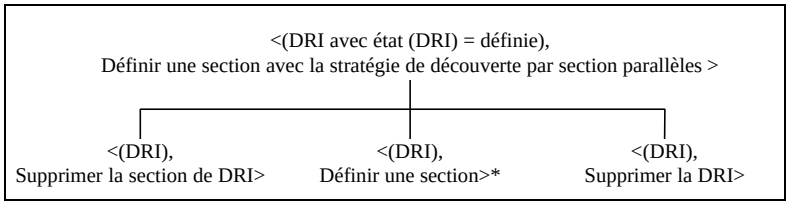


Figure 72: DRI10 : <(DRI avec état (DRI) = définie), Définir une section avec la stratégie de correction>

La définition d'une nouvelle section consiste à définir soit des sections ayant des stratégies alternatives si la DRI est de type choix soit des sections successives si la DRI est de type plan.

2.2.1.11 DRI11 : <(DRI, avec état (DRI) = définie), Identifier un composant avec la stratégie de découverte par section>

La DRI11 propose à l'ingénieur de méthodes de considérer chaque DRI de la carte de la méthode comme un composant de méthode potentiel. Ainsi il est nécessaire d'évaluer une DRI sélectionnée et de décider si elle est une directive représentant un module de processus réutilisable en dehors de sa méthode d'origine ou non. Si c'est le cas elle va être définie comme composant de méthode par la suite.

2.2.1.12 DRI12 : <({DRI, avec état (DRI) = définie}), Identifier un composant avec la stratégie de découverte par sections parallèles>

La DRI12, présentée à la Figure 73 par un plan, consiste à identifier deux ou plusieurs DRI appartenant à des sections parallèles dans la carte de la méthode. Les sections sont dites parallèles quand elles ont la même intention source et la même intention cible. Ensuite, l'ingénieur de méthodes doit décider si l'ensemble de ces directives peut être considéré comme un composant de méthode. Si c'est le cas, la directive parallélisme (DP) doit être créée à base de ces DRI afin de constituer le corps du nouveau composant de méthode.

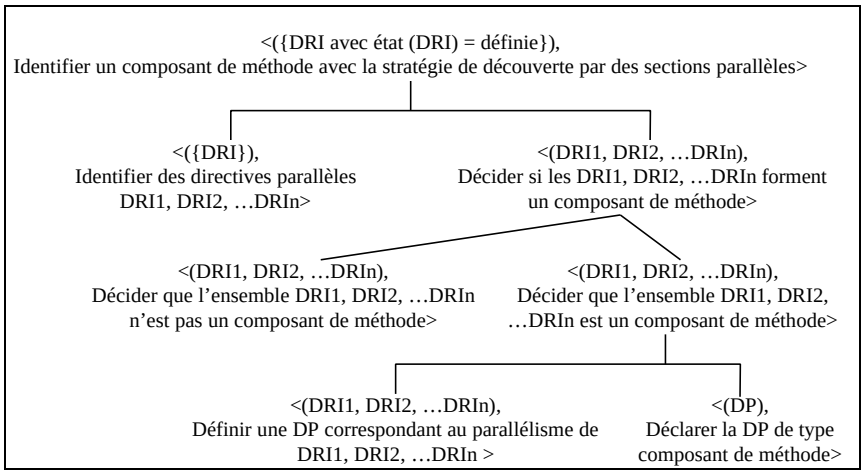


Figure 73 : DRI12 : <({DRI, avec état (DRI) = définie}), Identifier un composant avec la stratégie de découverte par sections parallèles>

La définition de la DP consiste à définir sa signature et à construire son corps (Figure 74). La signature de la DP doit être définie de la manière suivante : la situation doit comporter les mêmes parties de produit que toutes les DRI comportent, l'intention doit être également la même que celle des DRI mais elle n'a pas de paramètre manière. Puisque la DP est une directive de type choix proposant les DRI initiales comme des alternatives, la construction de son corps consiste à définir des arguments de choix pour chaque alternative (Figure 74).

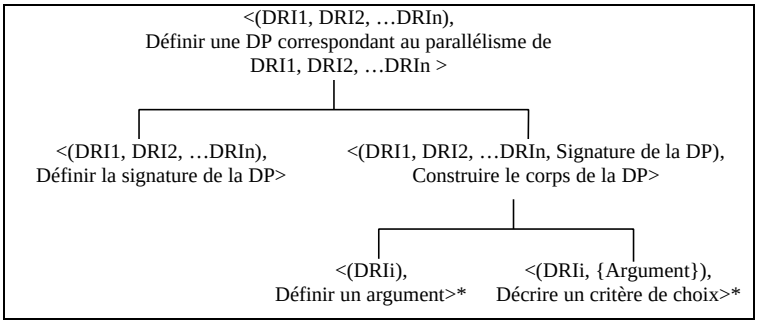


Figure 74 : <(DRI1, DRI2, ... DRIn), Définir une DP>

2.2.1.13 DRI13 : <({DRI, avec état (DRI) = définie}), Identifier un composant avec la stratégie de découverte par enchaînement de sections>

La DRI13, décrite à la

Figure 75 par un plan, propose d'identifier dans la carte de la méthode deux DRI qui s'enchaînent l'une après l'autre, c'est-à-dire que l'intention cible de la première section est la même que l'intention source de la deuxième section. Ensuite, l'ingénieur de méthodes doit décider si cet enchaînement peut donner droit à un composant de méthode. Si c'est le cas, il doit construire une *directive séquence* (DS) correspondant à cet enchaînement qui va représenter la directive de ce composant. C'est une directive plan qui comporte toutes les DRI d'enchaînement. Sa signature doit être définie de la manière suivante : la situation doit comporter les mêmes parties de produit que comporte la première DRI de l'enchaînement dans sa situation, l'intention doit être la même que celle de la dernière DRI dans l'enchaînement sans préciser le paramètre *manière*. Puisque la DS est une directive de type plan la construction de son corps consiste à définir le plan de précédence entre les DRI qui la composent (

Figure 76).

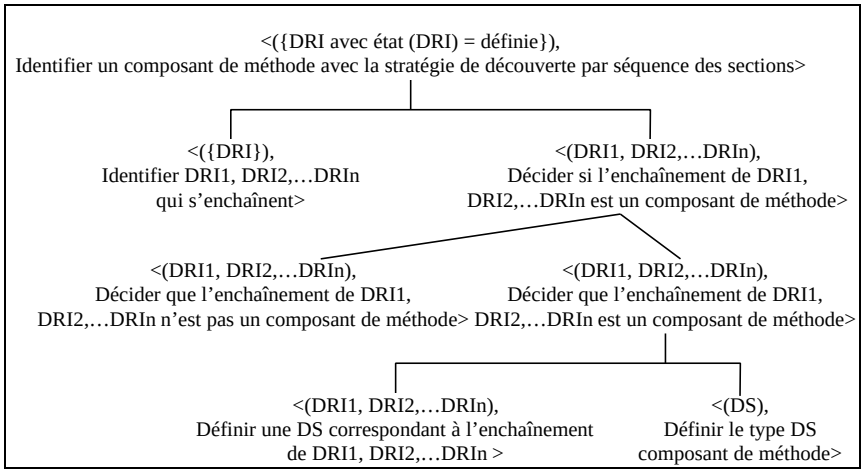


Figure 75 : DRI13 : <({DRI, avec état (DRI) = définie}), Identifier un composant avec la stratégie de découverte par enchaînement de sections>

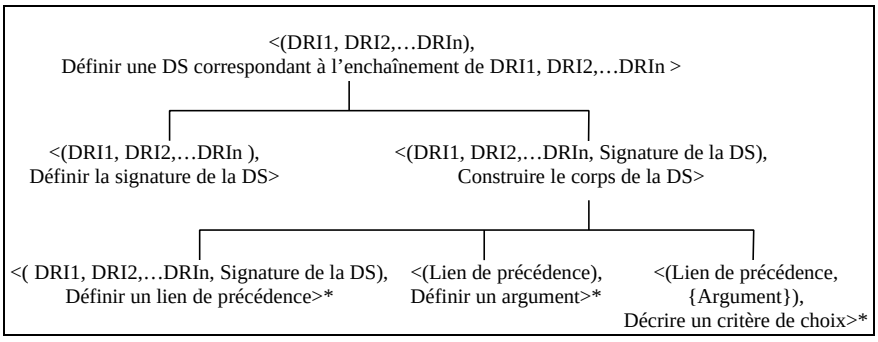


Figure 76 : <(DRI1, DRI2, ... DRIn), Définir une DS>

2.2.1.14 DRI14 : <(Composant, avec état (Composant) = identifié), Identifier un composant avec la stratégie de découverte par décomposition>

Après avoir identifié au moins un composant de méthode dont la directive est de type tactique ou stratégique, l'ingénieur de méthodes peut sélectionner la *stratégie de décomposition* dans la carte de ré-ingénierie (Figure 45) qui suggère de vérifier si ce composant n'est pas un composant agrégat. Un composant ayant une directive de type simple est toujours atomique. La DRI14 associée à cette section de la carte de ré-ingénierie est illustrée à la Figure 77. Suivant le type de la directive du composant initial elle propose différentes possibilités de décomposition de cette directive.

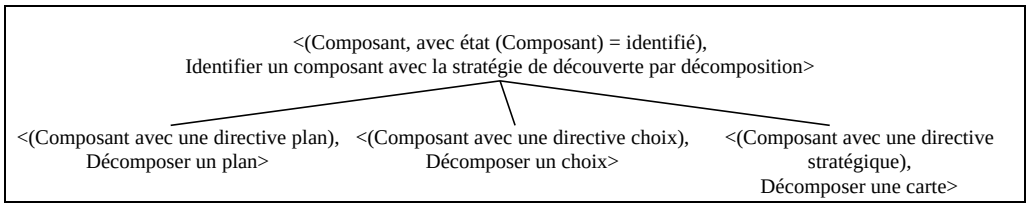


Figure 77 : DRI14 : <(Composant, avec état (Composant) = identifié), Identifier un composant avec la stratégie de découverte par décomposition>

Si la directive du composant est de type plan, la DRI14 suggère de vérifier si ces sous-directives pourraient être considérées comme des composants de méthode de niveau de granularité plus bas. Si c'est le cas, chacune de ces sous-directives est déclarée de type composant et le composant initial est déclaré de type agrégat (Figure 78). De même pour le cas d'une directive choix, où chaque alternative peut devenir un composant de méthode de niveau de granularité plus fin. Si la directive du composant est de type stratégique, c'est-à-dire représentée par une carte et des directives associées, le processus de décomposition applique de manière récursive le processus de découverte des composants proposé par la carte de ré-ingénierie de la Figure 45.

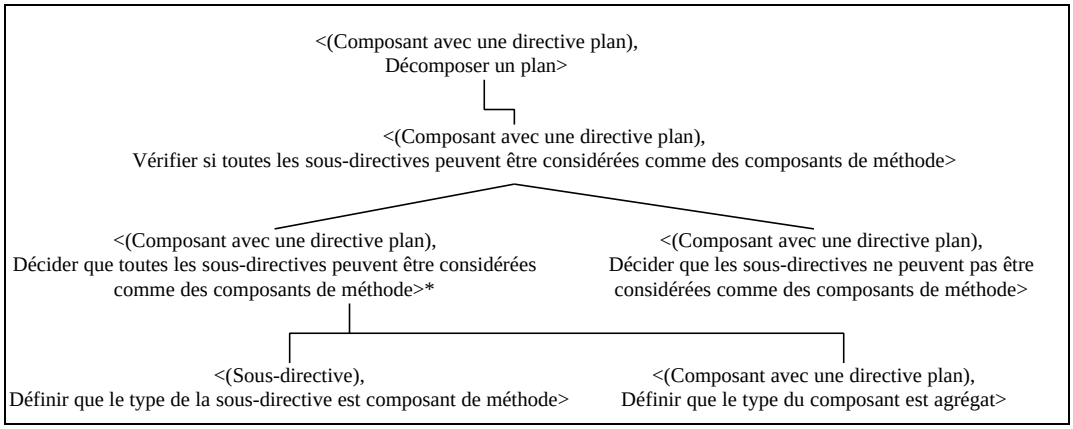


Figure 78 : <(Composant avec une directive plan), Décomposer un plan>

2.2.1.15 DRI15 : <({Composant, avec état (Composant) = identifié}), Identifier un composant avec la stratégie de découverte par agrégation>

La DRI16 permet à l'ingénieur de méthodes d'identifier des nouveaux composants à partir des composants identifiés précédemment. La directive propose deux possibilités pour agréger les composants : l'agrégation des composants représentants des démarches alternatives pour construire le même produit et l'agrégation des composants représentants des démarches pour construire des produits complémentaires.

La première possibilité consiste à identifier un nouveau composant à base des composants dont les signatures comportent les mêmes parties de produit dans leurs situations et les mêmes parties de produit comme cibles de leurs intentions. Bien sûr, leurs manières doivent être différentes. La directive du nouveau composant doit être définie comme une directive choix dont chaque alternative sélectionne un des composants initiaux.

La deuxième possibilité d'agrégation est basée sur la sélection des composants où la cible de l'intention d'un composant est la source du deuxième composant et ainsi de suite. La directive d'un tel composant est un plan dont chacune des sous-directives sélectionne un de ses sous-composants.

Le processus d'agrégation des composants est décrit à la figure ci-dessous.

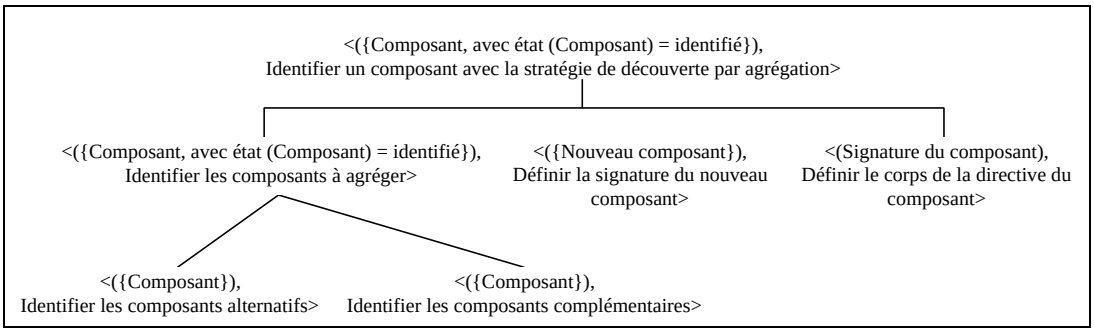


Figure 79 : DRI15 : <({Composant, avec état (Composant) = identifié}), Identifier un composant avec la stratégie de découverte par agrégation>

2.2.1.16 DRI16 : <(Composant, avec état (Composant) = identifié), Définir un composant avec la stratégie d'utilisation de formulaire>

La DRI16 aide l'ingénieur de méthodes à définir un composant de méthode à la base de la directive identifiée au préalable comme étant réutilisable en tant que composant de méthode. Elle propose un formulaire à remplir afin de définir le descripteur du composant ainsi que les parties de produit nécessaires à l'exécution de ce composant. Des valeurs prédéfinies sont proposées pour certaines cases du formulaire.

2.2.1.17 DRI17 : <(Composant, avec état (Composant) = identifié), Définir un composant avec la stratégie guidée>

La DRI17 présentée à la Figure 80, guide l'ingénieur de méthodes dans la définition du composant de méthode à partir de la directive déclarée précédemment comme un composant réutilisable. Cette définition consiste à sélectionner les parties de produit nécessaires à l'application de la directive dans le modèle de produit de la méthode et à définir le descripteur du composant.

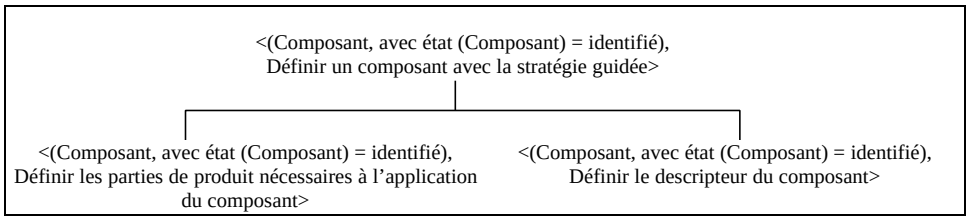


Figure 80 : <(Composant, avec état (Composant) = identifié), Définir un composant avec la stratégie guidée>

Suivant le méta-modèle de la méthode proposé au Chapitre 3, le descripteur a également la structure contextuelle, c'est-à-dire une situation et une intention. Ainsi, il est nécessaire de définir la situation de descripteur ce qui revient à l'identification du domaine (ou des domaines) dans lequelle le composant est applicable et à l'identification de l'activité de la conception qui pourrait utiliser le composant. L'intention de descripteur doit refléter l'objectif que le composant permet d'atteindre dans l'activité de la conception correspondante. Elle respecte la même structure que nous proposons dans le Chapitre 3 pour représenter toute intention de notre approche. En plus de la situation et de l'intention de réutilisation le descripteur décrit également d'une manière informelle l'objectif du composant, l'appartenance à une méthode avec des références vers la littérature présentant cette méthode. Il peut également comporter un exemple d'application. La Figure 81 exprime le processus de la définition d'un descripteur par une directive plan.

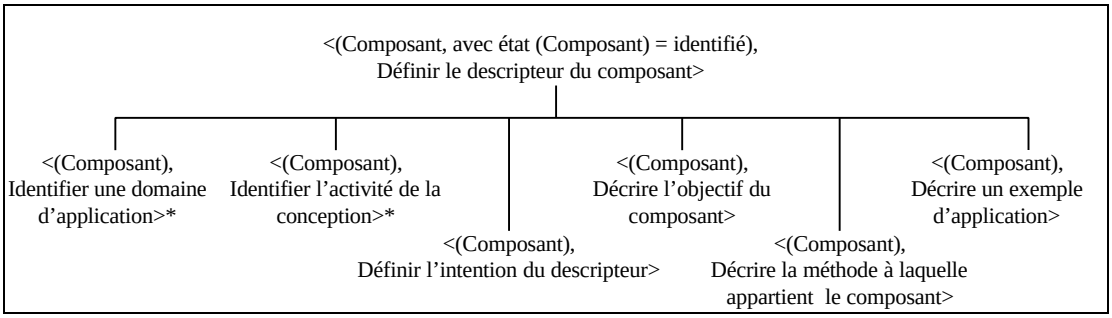


Figure 81 : <(Composant, avec état (Composant) = identifié), Définir le descripteur du composant>

2.2.1.18 DRI18 : <(Composant, avec état (Composant) = définie), Définir un composant avec la stratégie de complétude>

La DRI18 permet de compléter la définition d'un composant après avoir identifié son type : atomique ou agrégat, en le précisant dans la description du composant. Si le composant est de type agrégat, la

DRI18 permet de définir tous ses sous-composants en listant leurs signatures. Ceci est très utile lors de la sélection des composants dans la base de méthodes car permet, après avoir sélectionné un composant, de retrouver tous ses sous-composants. Dans le cas d'un composant atomique, la DRI18 permet de définir les composants agrégats qui comportent ce composant atomique et de lister leurs signatures dans la description de ce composant. Comme dans le cas précédent, ceci permet de retrouver un composant agrégat à partir d'un composant atomique faisant partie de cet agrégat.

2.2.1.19 DRI19 : <(Composant, avec état (Composant) = identifié), Arrêter avec la stratégie de vérification>

La DRI19 permet de terminer le processus de définition d'une méthode modulaire en vérifiant tout d'abord si toutes les directives de la carte de la méthode ont été définies. Ensuite, la DRI19 propose de vérifier si toutes les DRI de la carte de la méthode et toutes les combinaisons de ces DRI ont été considérées par le processus d'identification des composants. Finalement, elle propose de vérifier si tous les composants identifiés ont été décrits complètement.

2.2.2 Directives de Sélection d'Intention

La carte de la ré-ingénierie de méthodes contient un ensemble de Directives de Sélection d'Intention (DSI), une pour chaque intention de la carte, permettant de guider la progression de l'ingénieur de méthode dans la carte par la recherche d'intention. Elles sont décrites ci-dessous.

2.2.2.1 DSI2 : <(Section, avec état (Section) = définie), Progresser de Définir une section>

La DSI2, présentée à la Figure 82, est associée à l'intention *Définir une section* de la carte de processus de ré-ingénierie de méthodes (Figure 45). Elle aide l'ingénieur de méthodes à progresser dans la carte de ré-ingénierie après avoir réalisé cette intention. Comme le montre la Figure 82, l'ingénieur de méthodes peut soit définir une nouvelle section à partir des sections existantes, soit définir les DRI associées à ces sections.

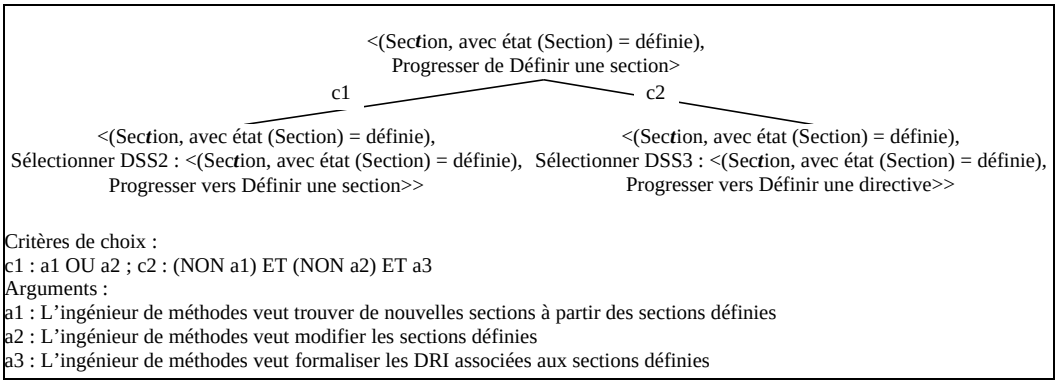


Figure 82: DSI2 : <{Section avec état (Section) = définie}, Progresser de Définir une section >

L'ingénieur de méthodes peut décider de réaliser l'intention *Définir section* s'il veut continuer à trouver de nouvelles sections ou s'il veut modifier les sections définies. Dans ce cas, il est invité à sélectionner

la DSS2 permettant de déterminer une stratégie pour définir une section. Par contre si l'ingénieur veut formaliser les DRI de la carte, il devrait sélectionner la DSS3 aidant l'identification d'une stratégie parmi les différentes possibles pour réaliser l'intention *Définir une directive*.

2.2.2.2 DSI3 : <(Directive, avec état (Directive) = définie), Progresser de Définir une directive>

Lorsque l'ingénieur de méthode a défini une directive de la carte de méthode, il peut soit corriger les sections définies si la description des directives a mis des erreurs en évidence soit, identifier des composants de méthode à partir des DRI définies. La DSI3 décrite à la Figure 83 guide l'ingénieur dans ce choix.

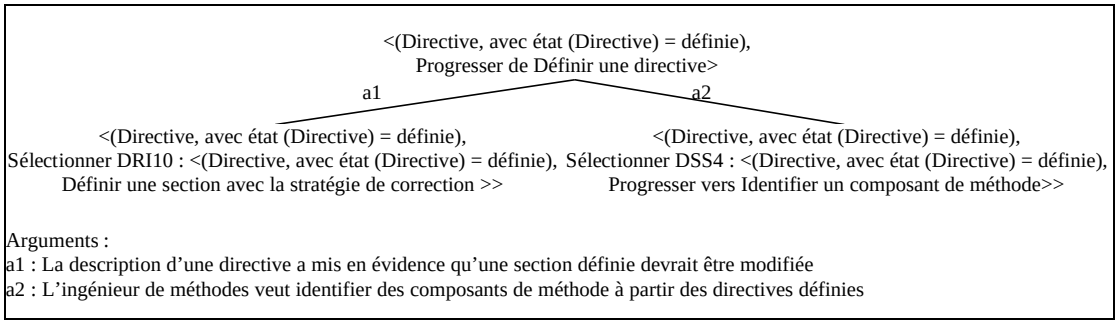


Figure 83: DSI3 : <(Directive avec état (Directive) = définie), Progresser de Définir une directive>

Si l'ingénieur de méthodes se rend compte d'une erreur impliquant une section, la DSI3 lui propose de sélectionner la DRI qui permet de corriger cette section. Sinon, il peut progresser vers l'identification des composants de méthode à partir des DRI qui sont déjà définies en sélectionnant la DSS4.

2.2.2.3 DSI4 : <(Composant, avec état (Composant) = identifié), Progresser de Identifier un composant>

2.2.3 Directives de Sélection de Stratégie

Pour tout ensemble de sections parallèles d'une carte il existe une Directive de Sélection de Stratégie (DSS) qui aide l'ingénieur de méthodes à choisir une stratégie parmi l'ensemble de stratégies proposées pour satisfaire l'intention cible en lui proposant des arguments de choix. Nous présentons par la suite les DSS associées à la carte de ré-ingénierie de méthodes.

2.2.3.1 DSS1 : <(Méthode avec état (Méthode) = modélisation initiale), Progresser vers Définir une section>

La DSI1 aide l'ingénieur de méthodes à démarrer le processus de ré-ingénierie en lui proposant deux possibilités : la *stratégie structurelle* et la *stratégie fonctionnelle*. Comme le montre la Figure 84, le choix de la stratégie dépend en partie de la présentation initiale de la méthode en considération. Si la méthode est décrite de manière informelle et que cette description concerne surtout la structure du

produit qu ‘elle propose, l’ingénieur de méthodes est invité à choisir la *stratégie structurelle* qui est basée sur l’analyse de cette structure. Il sélectionne alors la DRI1 décrite à la section 2.2.1.1. Par contre, si la description de la méthode prend aussi en compte le processus de la construction du produit et que ce processus est décrit de manière plus ou moins structurée, l’ingénieur de méthodes peut choisir la stratégie fonctionnelle qui est basée sur l’analyse de ce processus. Dans ce cas il sélectionne la DRI2 qui et présentée à la section 2.2.1.2.

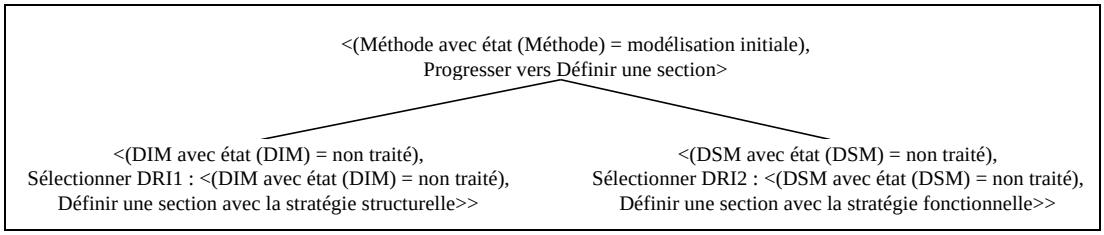


Figure 84 : DSS1 : <(Méthode avec état (Méthode) = modélisation initiale), Progresser vers Définir une section>

2.2.3.2 DSS2 : <(Section, avec état (Section) = définie), Progresser vers Définir une section>

Selon la carte de processus de ré-ingénierie de méthodes (Figure 45) il est possible de découvrir des nouvelles sections à partir d'un ensemble de sections déjà définies en suivant une parmi les quatre stratégies proposées: *stratégie de découverte par regroupement*, *stratégie de découverte par décomposition*, *stratégie de découverte d'alternative*, *stratégie de découverte par progression*. La DSS2, présentée à la Figure 85, guide l'ingénieur de méthodes dans la sélection d'une stratégie.

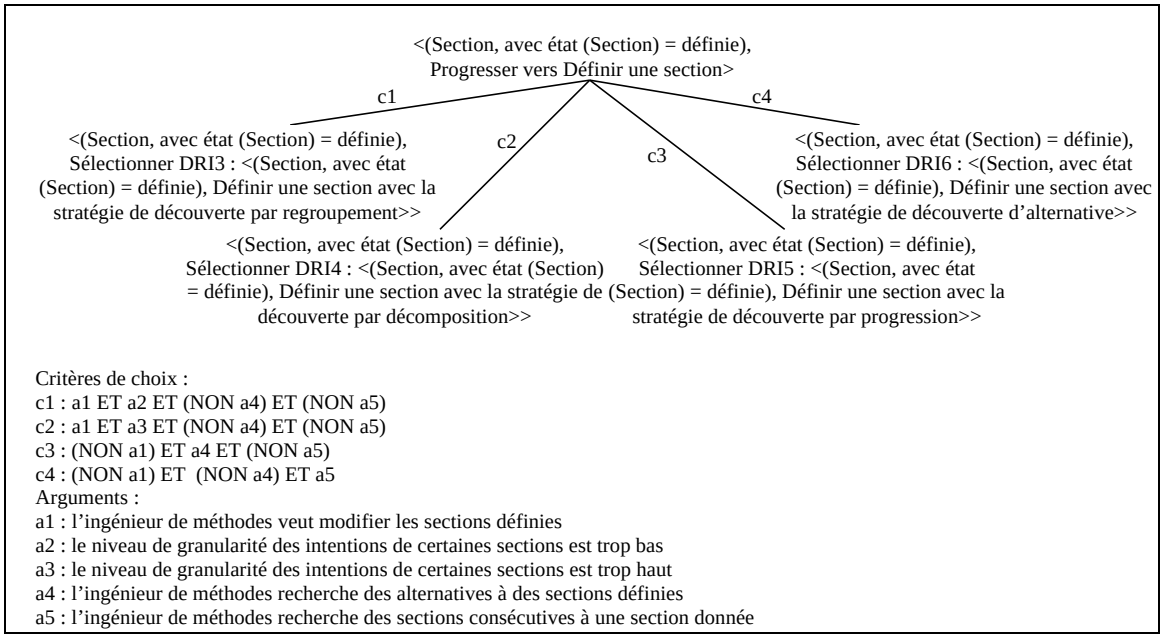


Figure 85: DSS2 : <{Section avec état (Section) = définie}, Progresser vers Définir une section>

Si l'ingénieur veut modifier les sections déjà définies et qu'il juge que certaines sections possèdent des intentions dont le niveau de granularité est trop élevé par rapport aux autres sections de la carte de méthode, la DSS2 lui propose de sélectionner la DRI3 permettant de définir une nouvelle section par regroupement. Par contre, si l'ingénieur juge que certaines sections possèdent des intentions dont le niveau de granularité est trop fin par rapport aux autres sections de la carte de méthode, il devrait sélectionner la DRI4 permettant de définir une nouvelle section par décomposition. Si l'ingénieur veut trouver des sections alternatives à des sections données, il est invité à sélectionner la DRI5 qui permet de le faire. Si l'ingénieur recherche des sections consécutives à des sections données, il devrait sélectionner la DRI6 permettant la définition de section dont l'intention source correspond à l'intention cible des sections données.

2.2.3.3 DSS3 : <(Section, avec état (Section) = définie), Progresser vers Définir une directive>

La progression vers la définition d'une directive est possible par une des trois stratégies proposées : la *stratégie d'utilisation de formulaire*, la *stratégie guidée* et la *stratégie de modification*. La DSS3 présentée à la Figure 86 aide l'ingénieur de méthodes à faire son choix parmi ces trois.

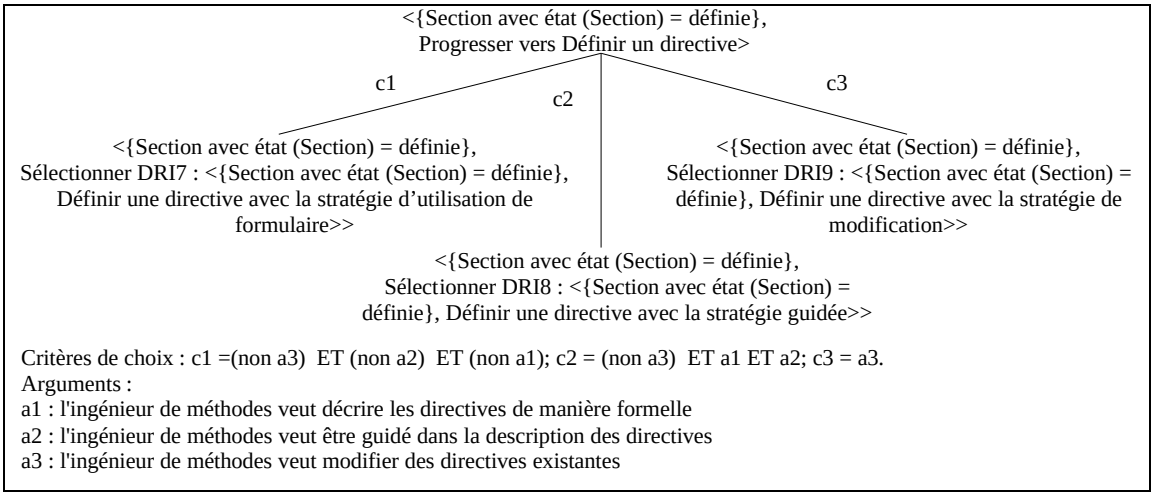


Figure 86: DSS3 <{Section avec état (Section) = définie}, Progresser vers Définir une directive>

Si l'ingénieur de méthodes veut décrire une directive de manière formelle, c'est-à-dire l'exprimer en tant qu'une directive exécutable, tactique ou stratégique, la DSS3 lui propose d'appliquer la stratégie guidée en sélectionnant la DRI8 proposée à la section 2.2.1.8. Par contre, s'il veut décrire une directive de manière informelle, la DSS3 lui suggère de choisir la stratégie d'utilisation de formulaire prise en compte par la DRI7 décrite à la section 2.2.1.7 de ce chapitre. La DRI7 fournit un formulaire de description des directives. Enfin, si l'ingénieur de méthodes veut décrire une directive en modifiant des directives existantes, il est invité à appliquer la stratégie de modification grâce à la DRI9 présentée à la section 2.2.1.9.

2.2.3.4 DSS4 : <(Directive, avec état (Directive) = définie), Progresser vers Identifier un composant>

Trois stratégies sont offertes à l'ingénieur de méthodes par le modèle de processus de ré-ingénierie pour progresser vers l'identification des composants de méthodes : la *stratégie de découverte par section*, la *stratégie de découverte par sections parallèles* et la *stratégie de découverte par enchaînement de sections*. La Figure 87 visualise la DSS4 qui aide l'ingénieur de méthodes à faire son choix parmi ces trois stratégies.

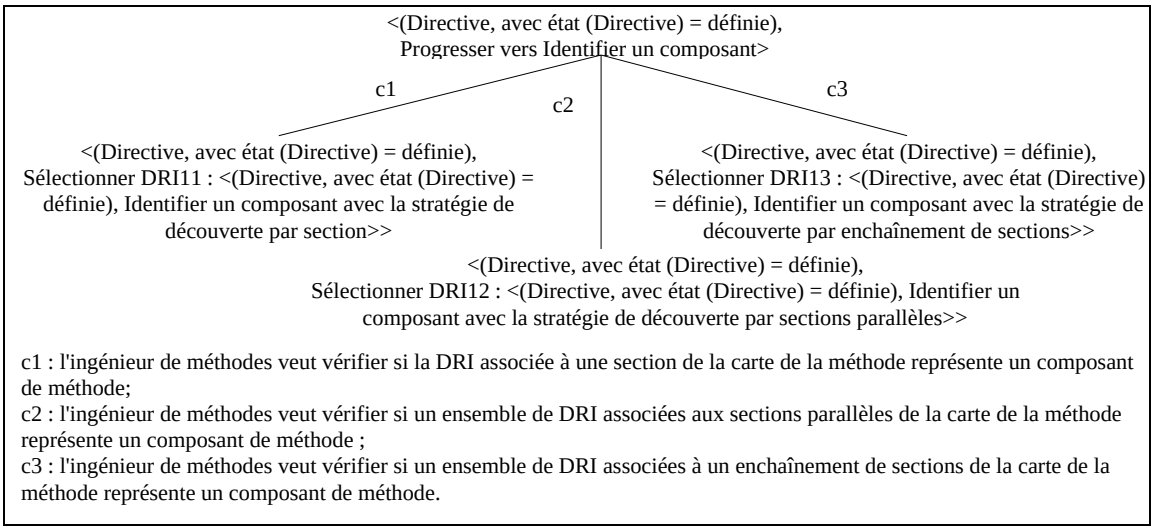


Figure 87 : DSS4 : <(Directive, avec état (Directive) = définie), Progresser vers Identifier un composant>

Le choix de la *stratégie de découverte par section* est basé sur la décision de l'ingénieur de méthodes d'analyser la carte de la méthode section par section en faisant hypothèse que chaque DRI associée à une section de la carte peut être considérée comme un composant de méthode. La DSS4 lui propose dans ce cas d'appliquer la DRI11 présentée à la section 2.2.1.11 de ce chapitre.

L'ingénieur de méthodes peut également décider de regrouper des différentes sections de la carte de la méthode en espérant que ce regroupement pourrait être considéré comme un composant de méthode. Le regroupement des sections parallèles est pris en compte par la DRI12 décrite à la section 2.2.1.12 qui suit la *stratégie de découverte par sections parallèles*. Le regroupement des sections qui s'enchaînent les unes après les autres est considéré par la DRI13 développée à la section 2.2.1.13 qui applique la *stratégie de découverte par enchaînement de sections*.

2.2.3.5 DSS5 : <(Composant, avec état (Composant) = identifié), Progresser vers Définir un composant>

La définition d'un composant identifié au préalable peut être réalisée en suivant une des deux stratégies proposées par le modèle de processus de ré-ingénierie : la *stratégie d'utilisation de formulaire* et la *stratégie guidée*. Le choix de la stratégie dépend surtout de l'expérience que l'ingénieur de méthodes a dans la définition des composants de méthode mais aussi de ses préférences.

Un expert n'a pas besoin d'être guidé, il peut choisir la *stratégie d'utilisation de formulaire* et profiter des valeurs prédéfinies proposées par celui-ci. Il sélectionne alors la DRI16 présentée à la section 2.2.1.16. Un novice dans la définition des composants peut choisir d'être guidé dans toutes les étapes de la description d'un composant. Il peut choisir la *stratégie guidée* et appliquer la DRI17 décrite à la section 2.2.1.7.

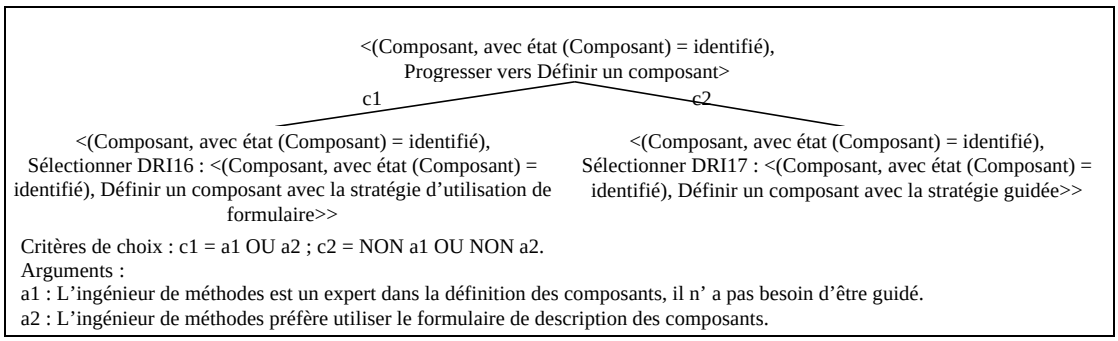


Figure 88 : DSS5 : <(Composant, avec état (Composant) = identifié), Progresser vers Définir un composant>

3. EXEMPLE DE RE-INGENIERIE

Nous utilisons la méthode OOSE pour illustrer l'application de notre modèle de processus de ré-ingénierie en utilisant comme source [Jacobson 92]. De plus nous ne prenons en compte que le modèle des cas d'utilisation proposés par cette méthode. La description source que l'on utilise est une description informelle décrivant la structure du modèle des cas d'utilisation en proposant quelques heuristiques telle que *"comment construire ce modèle"*. Par conséquent, on choisit la stratégie structurelle pour démarrer le processus de ré-ingénierie.

La DRI1, présentée à la Figure 47, propose d'identifier tout d'abord la signature de la carte de la méthode. La méthode peut être appliquée à partir d'une description informelle du problème, on identifie alors, *"Description du problème"* comme produit de la situation. L'objectif de la méthode est de construire un modèle des cas d'utilisation d'un système ce qui nous amène à identifier *"Construire le modèle des cas d'utilisation avec la stratégie de* OOSE*"* comme l'intention de la signature. En résumé, la signature de la carte de la méthode est la suivante :

<(Description du problème), Construire le modèle des cas d'utilisation avec la stratégie de OOSE>

Ensuite, la DRI1 propose d'identifier les intentions potentielles de la carte de la méthode. L'identification des intentions est basée sur l'identification des parties de produit fondamentales dans le produit de la méthode (Figure 49). Suivant la description proposée dans notre source, les parties de produit principales dans la structure d'un modèle des cas d'utilisation sont : des *acteurs*, des *scénarios* et des *cas d'utilisation*.

Une fois que l'on a identifié les parties de produit, on doit identifier les verbes représentant les manipulations qui doivent être effectuées sur celles-ci. Ces verbes doivent être sélectionnés dans le glossaire de la base de méthodes que nous proposons en Annexe. Supposons que l'on a sélectionné pour chaque partie de produit des verbes suivants :

Partie de produit	**Verbe d'intention**
Acteur	*Identifier, Définir*
Scénario de base	*Ecrire, Valider*
Scénario d'exception	*Ecrire, Valider*
Cas d'utilisation	*Identifier, Découvrir, Décrire, Conceptualiser*

En se basant sur ce tableau, on combine les verbes avec les parties de produit afin de construire des intentions (verbe + cible) possibles. On prend également en compte la description informelle de la méthode concernant ces parties de produit afin de vérifier si l'intention que l'on a identifié est traitée par celle-ci on non. Ainsi, on ne sélectionne que des intentions dont la réalisation est mentionnée dans la description de la méthode. Par exemple, la structure du concept *acteur* est très simple, il n'a que deux attributs : le *nom* et la *description informelle*. Ceci montre que l'intention *Définir acteur* n'a pas beaucoup d'intérêt car il n'est pas nécessaire de séparer le processus d'identification des acteurs de leur description. L'intention *Identifier acteur* peut suffire pour traiter ce concept.

Ce qui concerne les parties de produit *scénario de base* et *scénario d'exception*, le processus de validation n'est pas mentionné dans la description de la méthode. Par conséquent on élimine les intentions *Valider le scénario de base* et *Valider le scénario d'exception* de notre liste d'intentions.

Les intentions *Identifier un cas d'utilisation* et *Découvrir un cas d'utilisation* représentent la même chose, car les verbes *identifier* et *découvrir* sont des synonymes dans notre glossaire. La réalisation de l'intention *Décrire un cas d'utilisation* peut être inclue dans la réalisation de l'intention *Découvrir un cas d'utilisation.*

Finalement, on sélectionne les intentions suivantes :

- *Identifier un acteur*
- *Ecrire un scénario de base*
- *Ecrire un scénario d'exception*
- *Découvrir un cas d'utilisation et*
- *Conceptualiser un cas d'utilisation.*

L'étape suivante est d'identifier les stratégies potentielles pour réaliser ces intentions (Figure 51). Pour cela, il faut retrouver différentes manières proposées dans la description de la méthode pour satisfaire les intentions identifiées précédemment. Chaque manière identifiée peut être considérée comme une stratégie ou comme une tactique d'une stratégie. Le Tableau 1 résume le processus d'identification des stratégies.

Intention	Manière	Stratégie
Identifier un acteur	Poser la question : quelles sont les personnes qui utilisent une ou plusieurs des tâches principales du système ? Poser la question : quelles sont les personnes qui supervisent et qui maintiennent le système ? Poser la question : quels sont les dispositifs matériels périphériques qui font partie du domaine d'application et qui doivent être utilisés ? Poser la question : quels sont les systèmes avec lesquels le système doit interagir ?	***Stratégie de questions***
Ecrire un scénario de base	Décrire le scénario principal donnant la meilleure compréhension du cas d'utilisation	***Stratégie de cas normal***
Ecrire un scénario d'exception	Décrire le scénario décrivant une variante du scénario de base correspondant à un fonctionnement exceptionnel	***Stratégie de cas exceptionnel***
Découvrir un cas d'utilisation	Poser la question: quelles sont les principales tâches de chaque acteur? Poser la question: l'acteur aura-t-il à lire, écrire ou modifier une information du système? Poser la question: l'acteur devra-t-il informer le système des modifications extérieures? Poser la question: l'acteur souhaite-t-il être informé de modifications inopinées	***Stratégie de découverte par acteur***
Conceptualiser un cas d'utilisation	Regrouper le scénario normal avec les scénarios d'exception concernant un cas d'utilisation	***Stratégie d'intégration***
	Etendre un cas d'utilisation déjà complet pour modéliser des parties optionnelles du cas d'utilisation, des scénarios de remplacement complexes qui n'arrivent que rarement, etc.	***Stratégie d'extension***
	Extraire des descriptions des parties communes à plusieurs cas d'utilisation et les définir par des cas d'utilisation abstraits.	***Stratégie d'extension***

Tableau 1 : Identification des stratégies de la méthode OOSE

Pour finir la définition des sections il faut identifier les liens de précédence entre les intentions trouvées précédemment (Figure 52). Pour chaque intention et une stratégie donnée il faut identifier la situation dans laquelle l'intention peut être appliquée, c'est-à-dire les parties de produit nécessaires à la réalisation de l'intention suivant cette stratégie. Ensuite, il faut identifier l'intention qui est la source de construction de ce produit. Par exemple, pour satisfaire l'intention *Identifier un acteur* suivant la *stratégie de questions* le produit nécessaire est la *description initiale du problème*. Puisque c'est une partie de produit qui existe dès le départ, l'intention source est l'intention *Démarrer*. Pour pouvoir écrire un scénario de base il faut que le cas d'utilisation correspondant soit découvert auparavant. Ceci veut dire que l'intention *Ecrire un scénario de base* suit l'intention *Découvrir un cas d'utilisation*. L'écriture d'un scénario d'exception ne doit pas précéder l'écriture d'un scénario de base selon la description source de la méthode. La découverte des cas d'utilisation est basée sur les acteurs identifiés au préalable. Par conséquent, l'intention *Découvrir un cas d'utilisation* suit l'intention *Identifier un acteur*. L'extension et l'abstraction des cas d'utilisation consistent à conceptualiser d'autre cas d'utilisation à partir d'un cas conceptualisé auparavant. Ce qui veut dire que les stratégies d'extension et d'abstraction bouclent sur l'intention *Découvrir un cas d'utilisatio*n. Le résultat de l'identification des sections est illustré à la Figure 89.

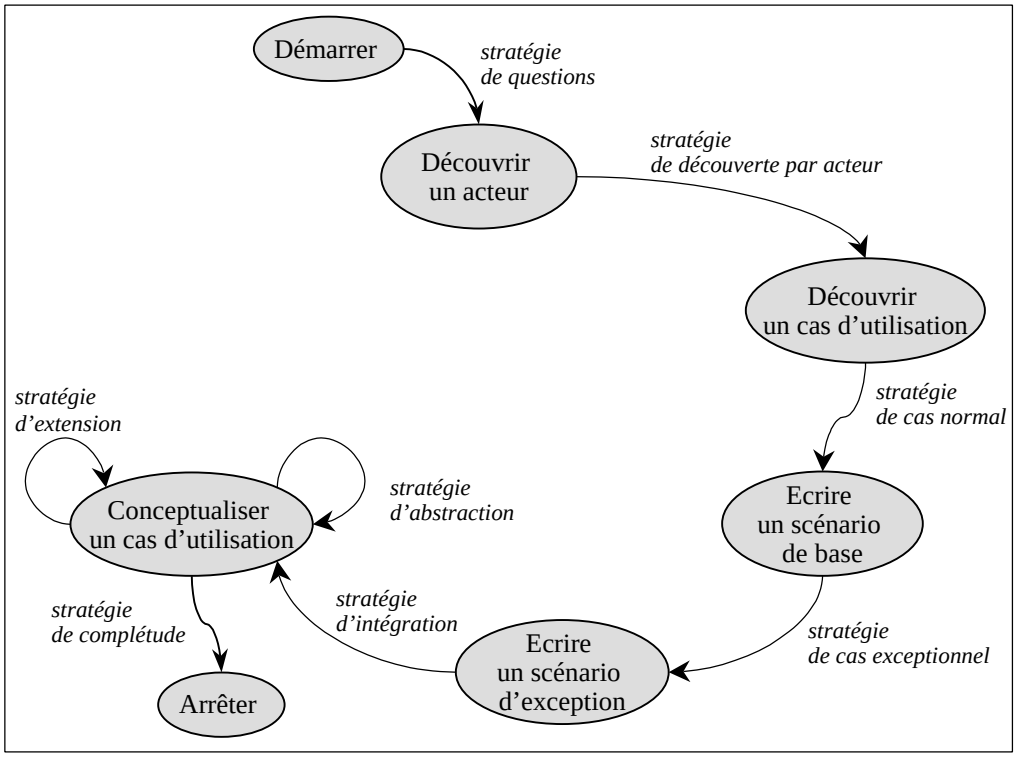


Figure 89 : Résultat initial d'identification des sections

Le résultat obtenu (Figure 89) nous semble trop linéaire ce qui nous amène à appliquer la stratégie de définition des sections *par regroupement*. La DRI3 réalisant cette stratégie (section 2.2.1.3) suggère de vérifier si certaines intentions ne pourraient pas être regroupées. Par exemple, dans la suite d'intentions *Découvrir un acteur, Découvrir un cas d'utilisation,* les deux intentions n'ont qu'une stratégie chacune. De plus, l'identification des acteurs ne sert qu'à la découverte des cas d'utilisation. On décide donc, de regrouper ces deux intentions et leurs stratégies. En d'autres termes, on intègre le

processus d'identification des acteurs dans le processus de découverte des cas d'utilisation et on obtient une section <*Démarrer, Découvrir un cas d'utilisation, Stratégie de découverte par acteur*>.

On procède de la même manière avec les intentions *Ecrire un scénario de base*, *Ecrire un scénario exceptionnel* et *Conceptualiser un cas d'utilisation*. Le guidage proposé par la description initiale de la méthode concernant l'écriture des scénarios et leur regroupement en cas d'utilisation est très pauvre. Ceci nous incite à regrouper ces intentions et leurs stratégies correspondantes et d'appeler la nouvelle stratégie la stratégie de cas normal d'abord. Le résultat de ce regroupement est la section <*Découvrir un cas d'utilisation, Conceptualiser un cas d'utilisation, Stratégie de cas normal d'abord*>.

Après avoir appliqué deux fois la stratégie de regroupement on obtient la carte illustrée à la Figure 90.

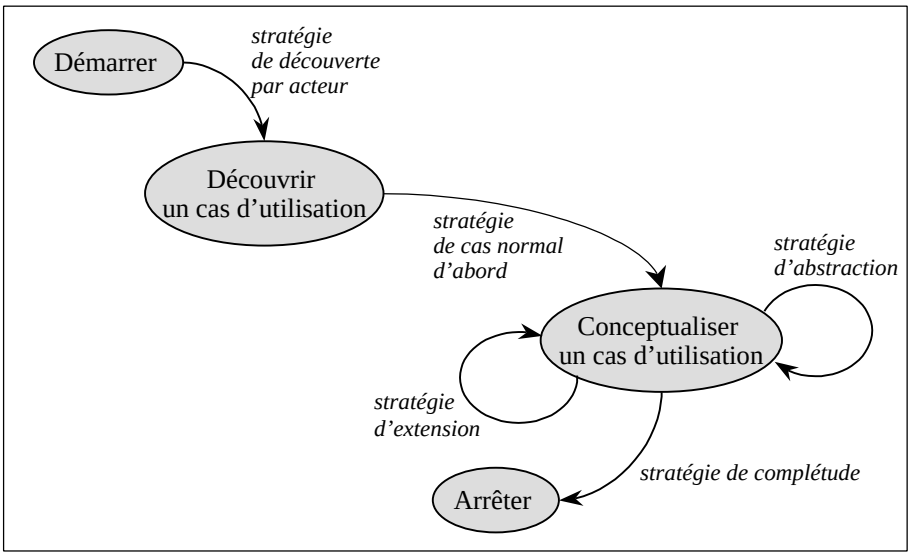


Figure 90 : Résultat après l'application de la stratégie de regroupement

En analysant cette carte, on s'aperçoit que les cas d'utilisation abstraits obtenus en appliquant la *stratégie d'abstraction* ne sont jamais utilisés par la suite. Pourtant, la description initiale de la méthode suggère de les réutiliser dans la description d'autres cas d'utilisation ayant les mêmes descriptions communes capturées dans les cas abstraits. Pour résoudre ce problème on choisit la *stratégie de découverte d'alternative* dans la carte de modèle de processus de ré-ingénierie (Figure 45). Cette stratégie est prise en compte par la DRI6 décrite à la section 2.2.1.6. Suivant la proposition de la DRI6, on définit une section alternative à la section <*Découvrir un cas d'utilisation, Conceptualiser un cas d'utilisation, Stratégie de cas normal d'abord*> en termes d'une stratégie alternative pour conceptualiser les cas d'utilisation. La nouvelle stratégie consiste à réutiliser les descriptions des *cas d'utilisation abstrait* dans l'écriture d'autre cas d'utilisation qui comportent les mêmes parties de description. Par conséquent, on ajoute une nouvelle section <*Découvrir un cas d'utilisation, Conceptualiser un cas d'utilisation, Stratégie de réutilisation*> dans la carte de la méthode et on obtient la carte qui est décrite à la Figure 91.

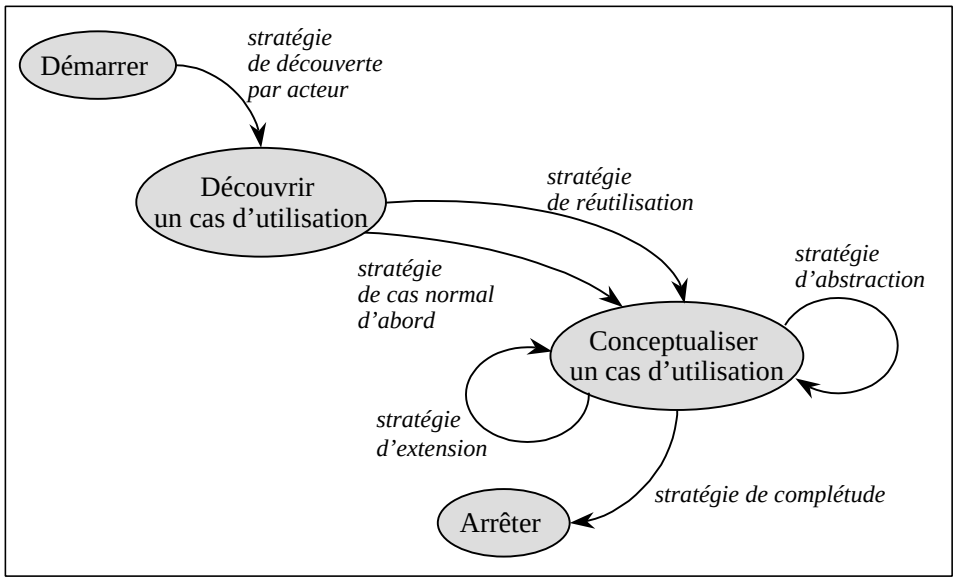


Figure 91 : Carte de la méthode OOSE

On passe maintenant à la définition des directives associées à la carte obtenue. On choisit la stratégie guidée qui est traitée par la DRI8 décrite à la section 2.2.1.8. Selon le méta-modèle de méthodes modulaires *(Chapitre3)*, toute section de la carte doit avoir une DRI associée décrivant comment réaliser l'intention cible de la section en suivant la stratégie de la section.

Prenons comme exemple la définition de la DRI pour la section *<Démarrer, Découvrir un cas d'utilisation, Stratégie de découverte par acteur>*. Comme le montre la Figure 62, la définition d'une DRI consiste à définir sa signature et son corps. La définition de la signature consiste à définir la situation dans laquelle la DRI peut être appliquée et l'intention qu'elle permet de satisfaire. La situation comporte dans la plupart des cas le produit cible de l'intention source de la section considérée. Dans notre cas, l'intention source est *Démarrer*. Le produit de la situation de la DRI en construction est donc la *description du problème*. L'intention d'une DRI correspond à l'intention cible de la section concernée avec un paramètre manière qui correspond à la stratégie de la section. Par conséquent, la signature de la DRI en construction est *<(Description du problème), Découvrir un cas d'utilisation avec la stratégie de découverte par acteur>*. La définition du corps de la DRI dépend de son type (simple, tactique ou stratégique). On considère que la DRI en construction est une directive tactique de type plan car elle comporte deux étapes principales : l'identification des acteurs et l'identification des cas d'utilisation. La construction du corps d'une directive plan est guidée par la Figure 66 et la description qui la suit. On identifie deux sous directives de la directive plan et on définit leur signature de manière habituelle :

- <(Description du problème), Définir un acteur>,
- <(Acteur), Définir un cas d'utilisation>.

Ensuite, on doit définir les liens de précédence entre ces sous directives et les argumenter. La définition des cas d'utilisation suit toujours la définition des acteurs car la première est basée sur la deuxième. Par contre, on peut définir les cas d'utilisation dès que l'on a défini un acteur ou bien après

avoir défini tous les acteurs. Ces deux manières de procéder représentent des arguments appelés *"le mode en largeur d'abord"* et *"le mode en profondeur d'abord"*. Pour compléter les critères de choix on doit également vérifier au fur et à mesure si tous les acteurs ont déjà été identifiés, si tous les cas d'utilisation d'un acteur ont été déterminés et si tous les cas d'utilisation de tous les acteurs ont été définis. Les différentes combinaisons de ces arguments nous permettent de définir le critère de choix de chaque lien de précédence. La Figure 92 montre la DRI obtenu.

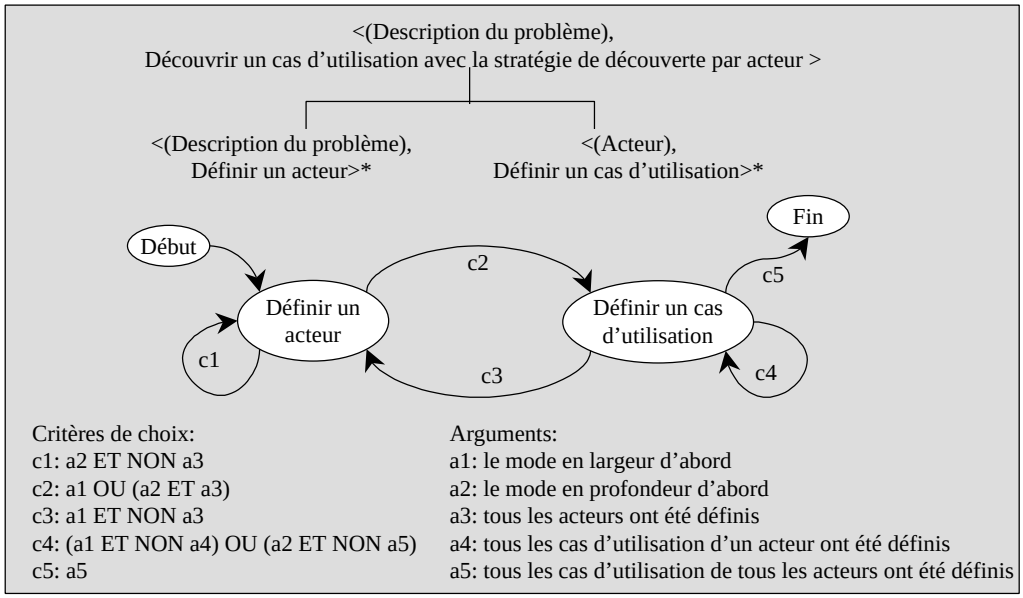


Figure 92 : DRI associée à la section <Démarrer, Découvrir un cas d'utilisation, Stratégie de découverte par acteur>

Suivant le même raisonnement on doit définir les corps des deux sous directives. Par exemple, la définition d'un acteur est une directive plan qui consiste à l'identifier puis à le décrire, c'est-à-dire lui donner un nom et une description informelle. Le même processus doit être exécuté pour définir un cas d'utilisation.

En appliquant la même stratégie, on peut définir les DRI de toutes les sections de la carte de notre méthode.

Il existe deux sections parallèles entre les intentions *Découvrir un cas d'utilisation* et *Conceptualiser un cas d'utilisation* dans la carte que l'on vient de construire (Figure 91). Cette situation nécessite une directive de sélection de stratégie (DSS). Pour définir cette directive nous appliquons également la stratégie guidée. La définition de sa signature suit la même procédure que celle que l'on a utilisé plus haut. Mis à part le fait que l'intention a une expression prédéfinie, son verbe est toujours *"Progresser"*. Par conséquent, la signature de la DSS considérée est la suivante :

> *<(Cas d'utilisation avec état (Cas d'utilisation) = identifié), Progresser vers Conceptualiser un cas d'utilisation>*

Comme nous l'avons déjà mentionné dans le Chapitre 3, une DSS est toujours une directive tactique de type choix. La DRI8 que l'on a utilisée pour définir la DRI précédente, aide également à définir une

directive choix. Ceci est décrit à la Figure 67 ainsi que la description qui la suit. Suivant cette directive on doit identifier les sous-directives de la directive choix qui représentent les différentes alternatives de réalisation de la directive initiale, argumenter chaque alternative et définir le corps de chaque sous directive. Puisque le rôle d'une DSS est d'aider à choisir une section parmi un ensemble de sections parallèles, ses sous-directives correspondent chacune à la sélection de la DRI associée à une section considérée. Le corps de notre DSS est illustré ci-dessous.

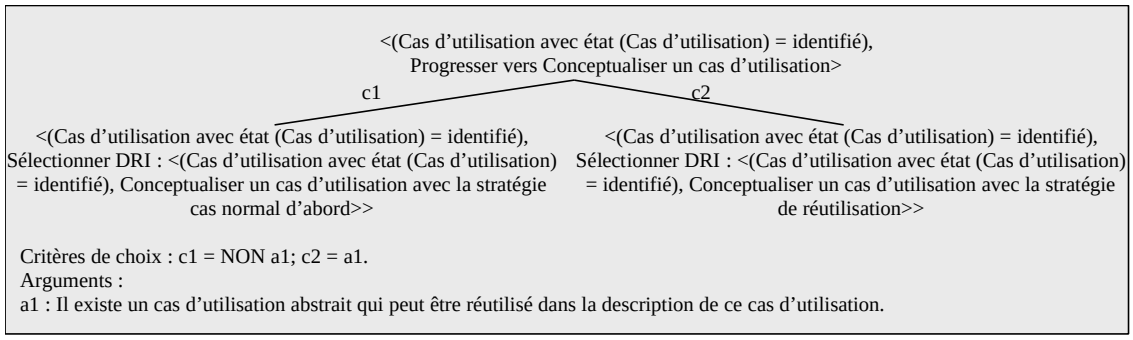


Figure 93 : Exemple d'une DSS associée à la carte de la méthode OOSE

La progression vers l'identification des composants de méthodes est conseillée après la définition de toutes les directives de la carte. Toutefois, il est possible d'identifier des composants dès la définition d'une DRI. On choisit d'abord la *stratégie de découverte par section* présentée à la section 2.2.1.11 pour identifier des composants de méthode. Suivant cette stratégie on vérifie pour chaque DRI de la carte de la méthode si elle représente un module de processus réutilisable. Par exemple, on peut considérer que la DRI associée à la section <*Démarrer, Découvrir un cas d'utilisation, Stratégie de découverte par acteur*>, décrite précédemment, est un composant de méthode car elle peut être appliquée chaque fois que l'on a besoin d'identifier des services qu'un système doit fournir à ses utilisateurs.

En choisissant la *stratégie de découverte par enchaînement de sections* décrite à la section 2.2.1.13 de ce chapitre, on considère que la découverte d'un cas d'utilisation suivie par sa conceptualisation peut être vue comme un composant de méthode. Pour cela, il faut définir la directive séquence qui représenterait le corps de ce composant. C'est un plan simple qui dit que l'exécution de la DRI de la section <*Démarrer, Découvrir un cas d'utilisation, Stratégie de découverte par acteur*> doit être suivie par l'exécution de la DRI associée à la section <*Découvrir un cas d'utilisation, Conceptualiser un cas d'utilisation, Stratégie cas normal d'abord*> de la carte de notre méthode. Nous illustrons cette directive à la Figure 94.

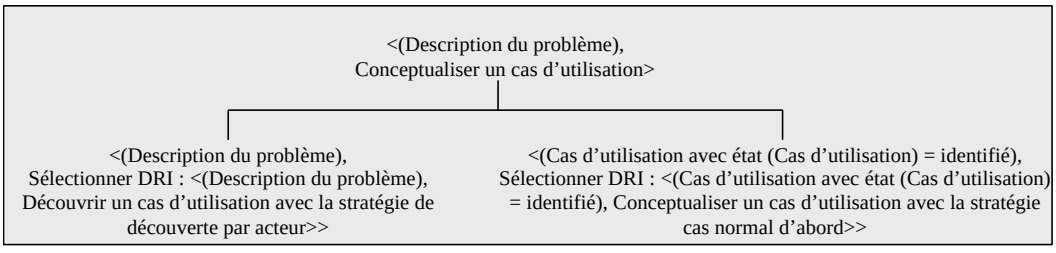


Figure 94 : Exemple d'une directive identifiée en tant que composant de méthode

Pour finir notre exemple nous illustrons la définition du composant de méthode présenté à la Figure 92 en appliquant la *stratégie guidée* décrite à la section 2.2.1.17. Suivant la DRI17 (Figure 80) on doit définir d'abord les parties de produit nécessaires à l'application de ce composant et ensuite définir son descripteur.

Les parties de produit utilisées pour la découverte des cas d'utilisation sont l'*acteur* et le *cas d'utilisation*.

Respectant la structure du descripteur proposée au Chapitre 3, on doit déterminer d'abord le domaine dans lequel le composant peut être appliqué. On peut retrouver cette information dans la description de la méthode correspondante car en général tous les composants d'une méthode s'appliquent dans les mêmes domaines d'application que la méthode elle-même. Sinon, on peut les évaluer en analysant le contenu du composant. Par exemple, le composant que l'on est en train de définir, peut être appliqué dans la conception des *systèmes d'information*, des *systèmes interactifs*, pour modéliser les *interfaces homme-machine* ou bien pour la *reconstruction des processus d'entreprise*. Ensuite, on doit définir laquelle des activités dans le processus de la conception est traitée par le composant. Par exemple, notre composant aide à réaliser l'activité de la *découverte des besoins* d'un système. Ceci est exprimé explicitement par l'intention de descripteur *Découvrir des besoins fonctionnels d'un système avec la stratégie des cas d'utilisation*. Ce sont les trois éléments principaux déterminant le contexte de réutilisation d'un composant. Pour faciliter la tâche de l'extraction des composants de la base il est conseillé de compléter cette information par la description informelle de l'objectif du composant. L'objectif de notre composant est d'aider l'ingénieur de développement à identifier les acteurs interagissant avec le système, que ce soit des utilisateurs humains à qui le système en construction est destiné, le personnel de maintenance ainsi que d'autres systèmes ou matériels qui utilisent ou sont utilisés par ce système. En plus de ça, le composant permet de découvrir les services que le système doit fournir à tous ses acteurs.

Il est également utile de définir des liens vers les sous-composants ou/et les super-composants d'un composant donné car ceux-ci permettent de faciliter la recherche et l'extraction des composants de la base. Par exemple, notre composant est un composant atomique, il n'a pas de sous-composants. Par contre, il est utilisé par d'autres composants que nous désignons par leurs signatures dans la section agrégats.

Finalement, il est conseillé de proposer un exemple d'application pour aider l'ingénieur d'applications à évaluer les capacités du composant.

La Figure 95 résume la présentation du composant de méthode que l'on vient de définir.

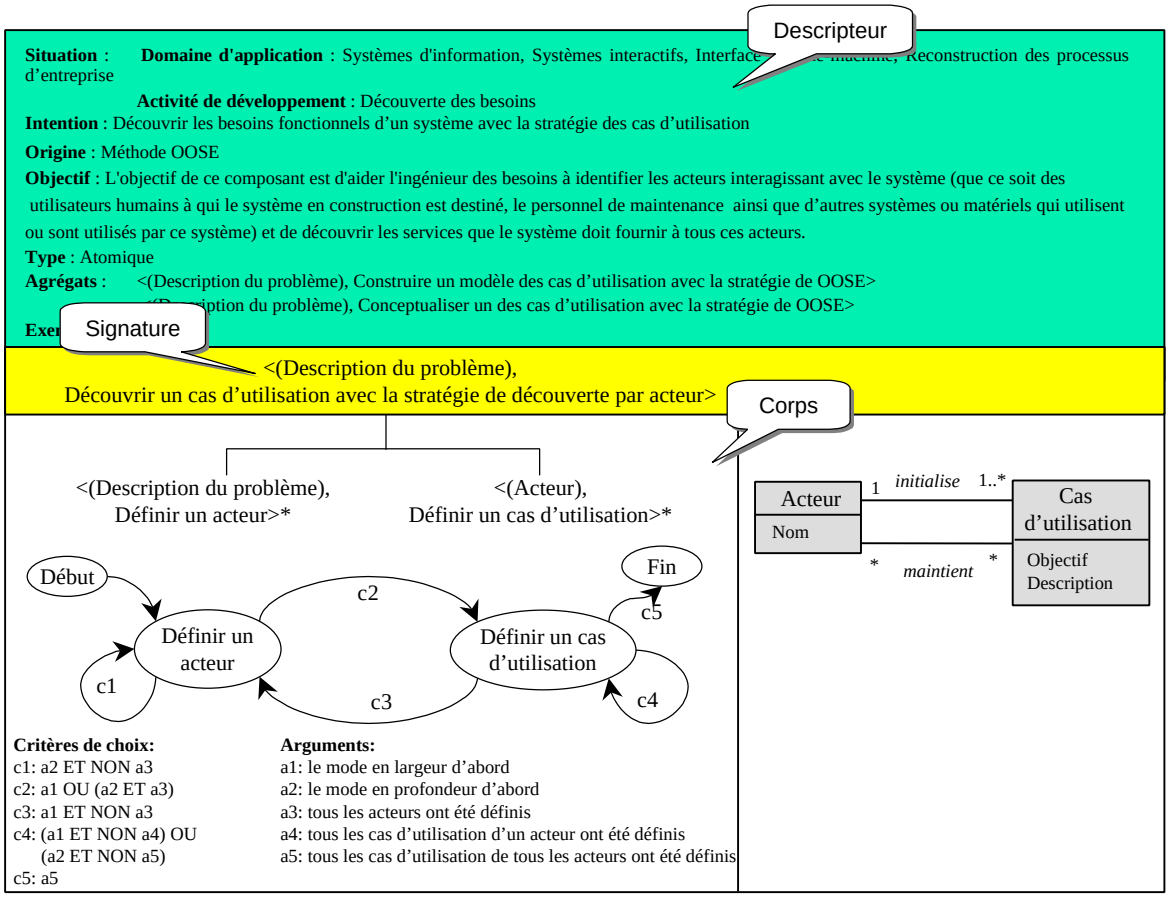


Figure 95 : Exemple d'un composant de méthode issu de la méthode OOSE

4. CONCLUSION

Dans ce chapitre nous avons proposé une approche d'ingénierie de méthodes qui permet de transformer des méthodes existantes en composants inter-reliés réutilisables dans la construction d'autres méthodes.

Il existe actuellement des approches proposant des différentes tactiques d'assemblage des composants de méthodes. Cependant, il n'y a aucune approche qui dit comment définir ces composants de méthodes, ni comment les stocker. Dans notre travail nous nous préoccupons de ce problème.

La solution que nous proposons est un modèle de processus de ré-ingénierie de méthodes qui peut être appliqué sur tous les types de méthode d'ingénierie de systèmes. Ce modèle de processus est basé sur le méta-modèle de méthodes modulaires que nous proposons dans le chapitre précédent.

Puisque notre modèle de processus de ré-ingénierie de méthodes est décrit sous forme d'une carte et de directives associées, à tout moment nous pouvons l'étendre et le compléter sans trop d'efforts par l'ajout de nouvelles stratégies ou /et de nouvelles intentions. La particularité de la carte est que c'est un modèle multi-démarches très flexible et facilement extensible. Ceci veut dire, qu'à chaque fois que l'on identifie un nouveau type de méthode, un nouveau type de directive ou un nouveau type de

composant, la carte de ré-ingénierie peut être complétée sans modifications importantes sur ce qui a déjà été défini.

PARTIE II

CONSTRUCTION DES METHODES PAR ASSEMBLAGE DE COMPOSANTS

Dans la partie précédente de ce mémoire, nous avons défini ce qu'est un composant de méthode et nous avons proposé une démarche pour redéfinir une méthode en termes de composants. Dans cette deuxième partie de notre travail, nous nous intéressons à l'assemblage de composants issus de différentes méthodes avec l'objectif d'en construire de nouvelles ou d'en enrichir des existantes.

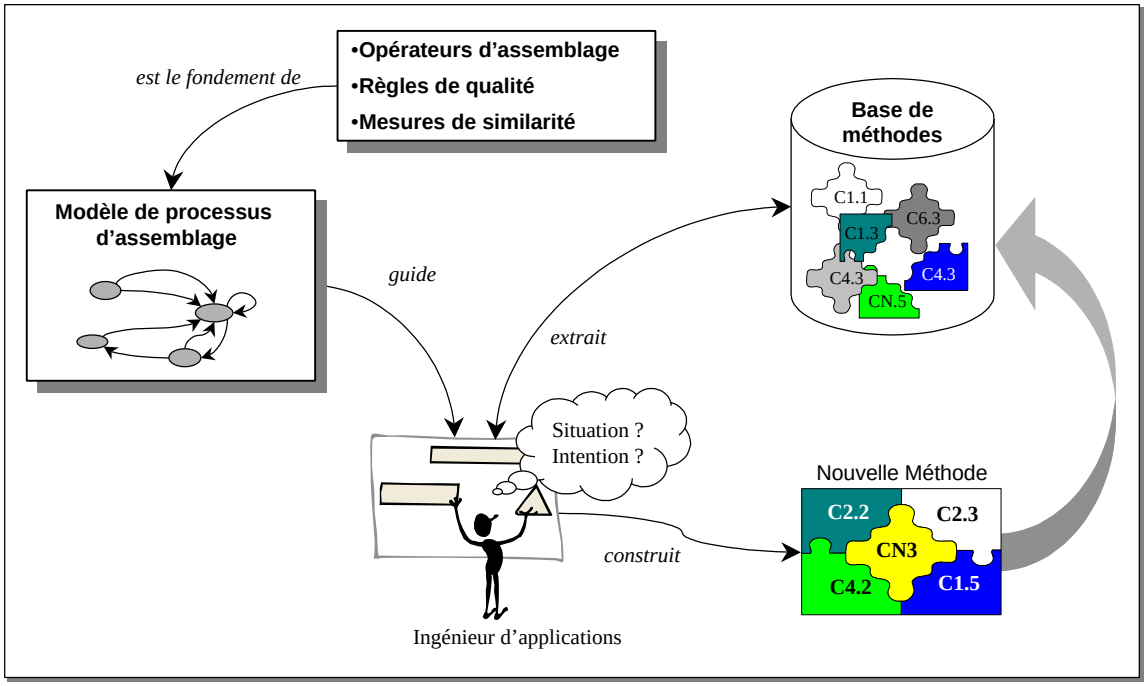


Figure 96 : Ingénierie des méthodes par assemblage des composants

La Figure 96 résume l'approche d'ingénierie des méthodes par assemblage de composants que nous proposons dans cette partie du mémoire. L'élément principal de cette approche est le modèle de processus d'assemblage que l'ingénieur d'applications applique dans le but de construire une nouvelle méthode à partir d'un ensemble de composants stockés dans une base. Ce modèle de processus guide l'ingénieur d'applications dans la sélection des composants et dans leur assemblage qui est fondé sur l'application des *opérateurs d'assemblage*, des *mesures de similarité* et des *règles de validation de la qualité* d'assemblage. La nouvelle méthode, le résultat du processus d'assemblage, peut à son tour être stockée dans la base de méthodes.

Les opérateurs d'assemblage permettent d'assembler les parties de produit des différents composants ainsi que leurs directives.

Notre approche d'assemblage, différemment des autres approches proposées actuellement, n'interdit pas l'assemblage des composants qui se couvrent partiellement, c'est-à-dire qui ont des éléments communs dans leurs parties de produit et/ou dans leurs directives. L'assemblage de tels composants nécessite de mesurer la similarité des différents éléments pour identifier éventuellement des éléments communs et de pouvoir choisir le mode d'assemblage adéquat. Par conséquent, nous avons introduit dans notre approche des mesures de similarités des éléments de modèles de produit et des éléments de modèles de processus.

Un autre élément faisant partie des fondements de notre approche est un ensemble de règles de validation permettant de vérifier si les opérateurs d'assemblage sont appliqués correctement et si la méthode obtenue est complète.

Ces trois éléments, décrits au Chapitre 5, sont sélectionnés et appliqués par le processus d'assemblage présenté au Chapitre 6.

CHAPITRE 5

Fondements de l'assemblage de composants de méthodes

L'assemblage de deux composants est basé sur l'assemblage de leurs parties de produit et de leurs directives respectives. Dans ce chapitre nous proposons un ensemble d'opérateurs qui sont utilisés par le processus d'assemblage des composants [Ralyte 99a]. Puisque nous devons assembler les modèles de produits et les modèles de processus il y a deux types d'opérateurs. Les opérateurs d'assemblage de produit permettent d'intégrer les modèles de produit des deux composants et les opérateurs d'assemblage de processus permettent d'assembler les directives des composants. Les deux types d'opérateurs sont présentés dans ce chapitre.

L'application de certains opérateurs nécessite de mesurer d'abord la similarité des éléments des composants à assembler. Ce chapitre comporte une section qui définit des mesures de similarité permettant de comparer la similarité sémantique et structurelle des éléments des parties de produit et des directives des composants de méthodes.

Finalement, la section 3 propose des règles permettant de vérifier l'exactitude de l'application des opérateurs et de valider la cohérence et la complétude de l'assemblage obtenu.

1. OPERATEURS D'ASSEMBLAGE

Dans cette section nous proposons des opérateurs qui sont utilisés par le processus d'assemblage de notre approche. Il y a deux types d'opérateurs, ceux utilisés dans l'assemblage des parties de produit et ceux servant dans l'assemblage des directives des composants. Dans les sous-sections suivantes nous

développons les deux types d'opérateurs. Tous les opérateurs sont présentés en appliquant un cadre qui contient des éléments suivants :

Motivation

Cette section contient la présentation informelle de l'opérateur en précisant à quoi il sert, quelles sont les motivations qui justifient son utilisation, quel est son objectif, etc.

Formalisme

Dans cette partie nous proposons une définition formelle de l'opérateur.

Démarche

Ici nous proposons une démarche à suivre pour appliquer l'opérateur. Cette démarche est exprimée sous forme d'une directive. Comme toute directive elle a une signature et un corps et elle appartient à un des trois types : simple, tactique ou stratégique.

Description

C'est l'explication informelle de la démarche d'utilisation de l'opérateur. Cette partie comporte également les conditions d'application de l'opérateur ainsi que le résultat qui doit être obtenu en appliquant l'opérateur.

Exemple

Dans cette partie nous proposons un exemple d'application de l'opérateur.

1.1 Opérateurs d'assemblage des modèles de produit

Dans notre approche nous proposons un certain nombre d'opérateurs permettant d'assembler les modèles de produit de différents composants de méthodes. Dans les sous-sections suivantes nous définissons d'abord la terminologie utilisée pour définir les opérateurs d'intégration. Ensuite, nous présentons les mesures de similarité des éléments de différents modèles de produit, puis la définition des opérateurs d'assemblage des modèles de produit et des règles de cohérence et de complétude du modèle intégré.

1.1.1 Méta-concepts des parties de produit

Dans cette section nous définissons les méta-concepts utilisés pour définir les opérateurs d'intégration des modèles de produit:

- *C* est un ensemble de concepts c_k, permettant de définir les objets du monde réel (des entités, des objets, etc…). Un concept est défini par un label unique c_k et il a un nom. Par exemple, *c_1 :*

Scénario et c_2 *: But* désignent deux concepts d'un modèle de produit. Nous pouvons aussi désigner le concept seulement par son label. La définition précise selon la forme de Bacchus est la suivante :

```
<concept> ::= <label_concept> : <nom_concept>
<label_concept> ::= Label ordonné, commençant par C et suivi d'un
nombre en index
<nom_concept> ::= Nom du concept
```

- *L* est un ensemble de liens l_k entre les concepts (des associations, des compositions, des spécialisations/généralisations). Un lien est défini par un label unique, un nom, un ensemble de labels et de noms des concepts qu'il relie aussi qu'un type de lien. Par exemple, l_1 *: composé de (*c_1 *: Scénario ,* c_2 *: Action, composition)* représente un lien de composition entre le concept *"Scénario"* et le concept *"Action"*. Le lien peut aussi être référencé seulement par son label.

```
<lien> ::= <label_lien> : <nom_lien> (<concept>, <concept>,
type_lien)
<label_lien> ::= Label ordonné, commençant par l et suivi d'un
nombre en indexe
<nom_lien> ::= Nom du lien
<type_lien> :: = association | composition |
spécialisation/généralisation
```

- *P* est un ensemble de propriétés structurelles p_k (des attributs). Une propriété a un label unique, un nom, le label et le nom du concept (ou de lien) auquel elle est attachée et un domaine de valeurs. Par exemple, p_m *: Objectif (*c_n *: Cas d'utilisation, chaîne de caractères)*. La propriété peut aussi être désignée seulement par son label.

```
<propriété> ::= <label_propriété> : <nom_propriété> (<concept>
|<lien> , <domaine>)
<label_propriété> ::= Label ordonné, commençant par p et suivi
d'un nombre en index
<domaine> ::= Domaine d'application
```

MP est un ensemble de modèles de produit pm_k où $MP \subseteq C*L*P$.

1.1.2 Typologie d'opérateurs d'assemblage de modèles de produit

Dans cette section, nous présentons les opérateurs d'assemblage des parties de produit. Nous proposons cinq types d'opérateurs pour assembler les parties de produit:

1. Les opérateurs d'*addition* permettant d'ajouter de nouveaux éléments à un modèle de produit en cours de construction.

2. Les opérateurs de *suppression* permettant d'enlever certains éléments du modèle de produit en cours de construction.

3. Les opérateurs d'unification permettant de changer les noms des différents éléments des modèles de produit utilisés dans l'assemblage.

4. Les opérateurs de *modification* destinés à modifier les modèles de produit assemblés en changeant le type de certains de leurs éléments.

5. Les opérateurs de *fusion* permettant de fusionner les éléments de deux modèles de produit différents en un nouvel élément du modèle en cours de construction.

1.1.2.1 Opérateurs d'addition

Dans cette section nous présentons les opérateurs d'intégration des modèles de produit permettant de compléter les modèles de produit par de nouveaux éléments. L'addition de nouveaux éléments peut être nécessaire dans l'assemblage de deux modèles de produit, pour faire le lien entre deux modèles n'ayant aucun élément commun. On peut être amené à ajouter de nouveaux concepts ou de nouveaux liens. L'addition des nouvelles propriétés peut être utile pour unifier les concepts de modèles de produit différents avant leur assemblage.

1.1.2.1.1 AJOUTER_CONCEPT

Motivation

Dans le cas où deux modèles de produit à assembler n'auraient aucun concept commun, il est parfois nécessaire de créer un nouveau concept permettant de faire le lien entre ces deux modèles. L'opérateur `AJOUTER_CONCEPT` permet de créer un nouveau concept et de l'ajouter dans le modèle de produit en cours de construction.

Formalisme

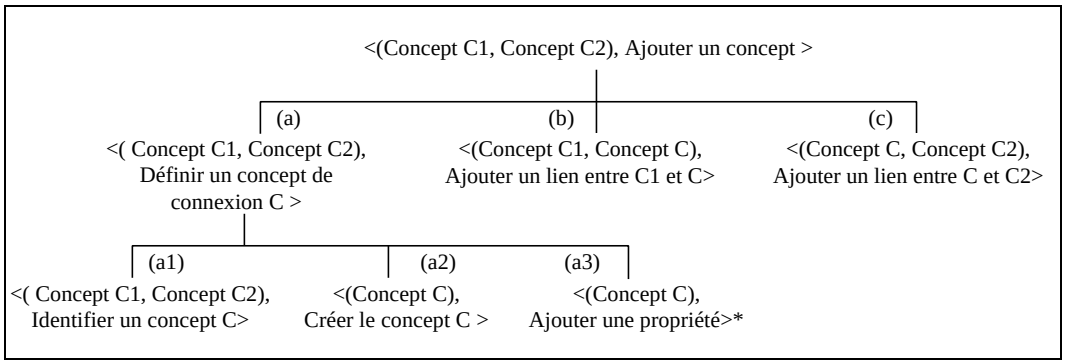
$$AJOUTER\_CONCEPT : C * C \rightarrow C$$

$$AJOUTER\_CONCEPT(c_{1j}, c_{2k}) = c_i \cup l_{ij}(c_i, c_{1j}) \cup l_{ki}(c_{2k}, c_i)) \mid c_{1j} \in mp_1 ; c_{2k} \in mp_2 ; c_i \in mp_{as} ; l_{ij} \in mp_{as} ; l_{ki} \in mp_{as}$$

Démarche

Signature : <(Concept C1, Concept C2), Ajouter un concept>

Corps:



Description

Un nouveau concept ne peut être ajouté que pour connecter deux modèles de produit n'ayant aucun élément commun. Le nouveau concept ajouté au moment d'assemblage n'a de sens que dans le modèle de produit en construction.

L'ajout d'un nouveau concept consiste à identifier le concept (a1), le créer (a2), c'est-à-dire lui donner un nom et à définir ses propriétés avec leurs domaines de valeurs (a3), enfin à l'associer aux concepts des deux modèles de produit (b et c). Par conséquent, au moins deux nouveaux liens doivent être ajoutés pour relier le nouveau concept aux concepts des deux modèles initiaux.

Exemple

Supposons que nous ayons à assembler deux composants de méthode tel que le premier composant propose une démarche pour écrire des scénarios et le deuxième propose une démarche pour construire un diagramme de classes. Pour assembler leurs parties de produit nous introduisons un nouveau concept appelé *"Ressource"*. Nous connectons ce concept avec le modèle de produit du premier composant en ajoutant un nouveau lien *"décrit"* et avec le modèle de produit du deuxième composant grâce au lien *"représenté par"* comme le montre la Figure 97. Ce nouveau concept permet de faire le lien entre les deux modèles de produit n'ayant aucun concept commun. Il permet d'identifier les ressources du système décrites dans les scénarios (utilisées dans le premier composant) qui deviennent des classes d'objets dans le diagramme de classes du deuxième composant.

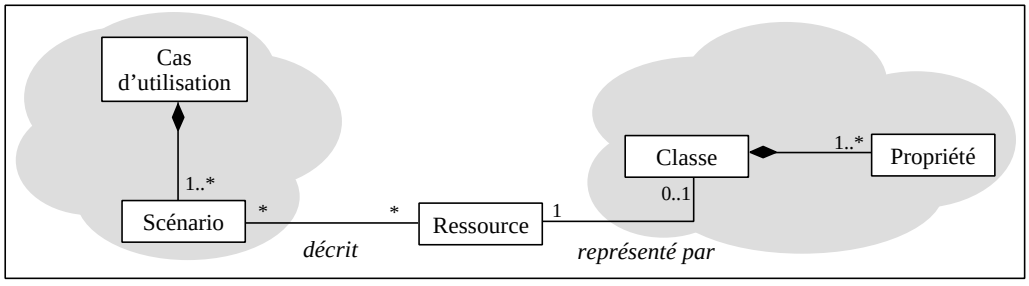


Figure 97 : Exemple d'application de l'opérateur AJOUTER_CONCEPT

1.1.2.1.2 AJOUTER_LIEN

Motivation

De la même manière que l'opérateur précédent, l'opérateur AJOUTER_LIEN sert à connecter deux modèles de produit qui n'ont aucun élément commun. Un lien de connexion peut suffire pour associer les éléments des deux modèles de produit assemblés.

Formalisme

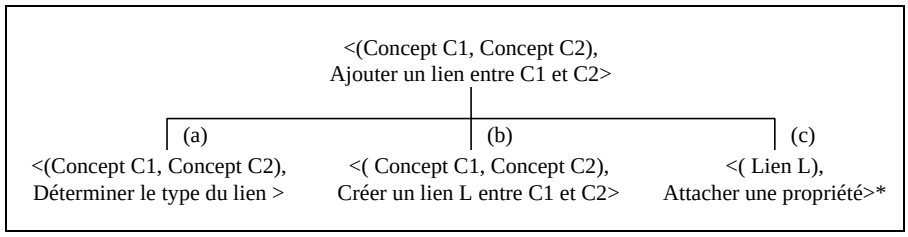
$AJOUTER\_LIEN : C * C \rightarrow L$

$AJOUTER\_LIEN(c_{1i},c_{2j}) = l_{ij}(c_{1i},c_{2j}) \mid c_{1i} \in mp_1 ; c_{2j} \in mp_2 ; l_{ij} \in mp_{as}$

Démarche

Signature : <(Concept C1, Concept C2), Ajouter un lien>

Corps :



Description

L'opérateur AJOUTER_LIEN permet de créer un nouveau lien entre deux concepts déjà existants. Le lien peut être une association, un lien de composition ou bien un lien de spécialisation / généralisation. Il est nécessaire que les concepts source et cible du lien existent déjà dans les modèles de produit avant la création du lien entre eux. Sinon il est nécessaire d'appliquer l'opérateur AJOUTER_CONCEPT pour créer les concepts manquants.

L'addition d'une nouvelle association ou d'un nouveau lien de composition ou nouveau lien de spécialisation / généralisation n'est possible que pour connecter deux concepts définis dans deux modèles différents. Le nouveau lien peut aussi connecter le concept créé au cours du processus d'intégration avec un concept d'un des modèles initiaux.

L'addition d'un nouveau lien consiste à déterminer le type du lien (a), à créer le lien (b) et à lui attacher des propriétés (c).

Exemple

Supposons que nous ayons à assembler un composant de méthode guidant l'écriture des scénarios avec un composant de méthode destiné à la construction des diagrammes de séquence, l'objectif étant de représenter chaque scénario par un diagramme de séquence. Comme le montre la Figure 98, l'addition d'un nouveau lien *"représenté par"* entre le concept *"Scénario"* du premier composant et le concept *"Diagramme de séquence"* du deuxième composant permet de faire la connexion entre les deux composants.

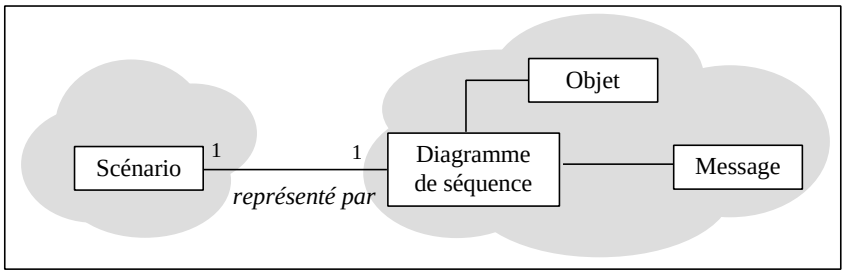


Figure 98 : Exemple d'application de l'opérateur AJOUTER_LIEN

1.1.2.1.3 AJOUTER_PROPRIETE

Motivation

L'addition de nouvelles propriétés permet d'enrichir le concept (ou le lien) auquel on les attache, de modifier sa structure et de l'adapter à la situation en cours.

Formalisme

AJOUTER_PROPRIETE : C $\rightarrow$ P

AJOUTER_PROPRIETE(c_i)= $p_{ik}(c_i)| \ c_i \in$ mp

ou

AJOUTER_PROPRIETE : L $\rightarrow$ P

AJOUTER_PROPRIETE(l_{ij})= $p_{ijk}(l_{ij})| \ c_i \in$ mp

Démarche

Signature : <(Concept C), Ajouter une propriété>

ou

Signature : <(Lien L), Ajouter une propriété>

Corps :

Actions :

1. Choisir un label de propriété
2. Choisir un nom de propriété
3. Définir le domaine de valeurs de la propriété

Description

L'opérateur AJOUTER_PROPRIETE permet d'attacher une nouvelle propriété à un concept ou à un lien déjà existant. Il est représenté par une directive exécutable qui comporte trois actions. L'ingénieur

de méthodes, tout d'abord, doit attribuer un label unique à la propriété, ensuite lui donner un nom et finalement définir le domaine de ses valeurs.

Exemple

Chaque fois que l'on ajoute un nouveau concept au modèle de produit en construction il est nécessaire de définir ses propriétés structurelles. Par exemple, l'addition d'un nouveau concept c_i *: "Ressource"* (Figure 97) nécessite l'addition de propriétés à ce concept. L'application de l'opérateur conduit aux deux propriétés : p_{i1} *: Nom_ressource (*c_i *: Ressource, char)* et p_{i2} *: Type_ressource (*c_i *: Ressource, {Données, Matériel}).*

1.1.2.1.4 SPECIALISER

Motivation

Cet opérateur correspond au cas où les deux modèles de produit à assembler contiendraient deux concepts qui ont une sémantique similaire, mais des structures différentes. Par exemple, le deuxième concept est plus riche que le premier et il contient toutes les propriétés du premier. Afin de permettre la cohabitation des deux concepts dans le modèle de produit final, il est possible de spécialiser le deuxième concept par rapport au premier. Un lien de spécialisation fait dans ce cas la connexion entre les modèles de produit initiaux.

Formalisme

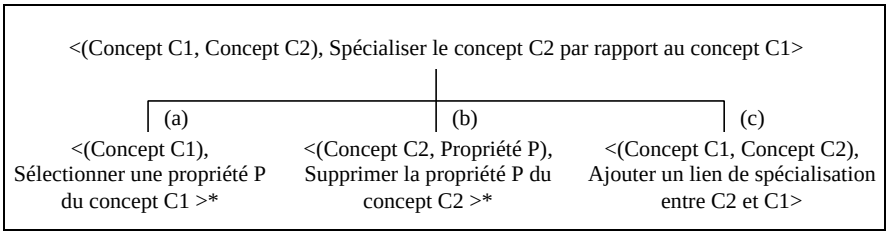
$\texttt{SPECIALISER} : C * C \rightarrow L$

$\texttt{SPECIALISER}(c_{1i}, c_{2j}) = l_{ij}$: est un$(c_{1i}, c_{2j})$; $c_{1i} \in mp_1$; $c_{2j} \in mp_2$; $l_{ij} \in mp_{as}$

Démarche

Signature : <(Concept C1, Concept C2), Spécialiser le concept C2 par rapport au concept C1>

Corps :



Description

L'opérateur SPECIALISER permet de définir un lien de spécialisation entre deux concepts appartenant aux modèles de produit différents. Les deux concepts doivent avoir la même sémantique mais être différents de point de vue structurel. Le premier doit avoir une structure plus simple que le

second, c'est-à-dire qu'il doit avoir moins de propriétés structurelles. De plus, le second doit contenir toutes les propriétés du premier. On appelle le premier concept le concept générique et le deuxième le concept spécifique. Le lien de spécialisation entre ces deux concepts fait la connexion entre les deux modèles de produit initiaux.

Le concept, déclaré spécifique, doit avoir au départ toutes les propriétés du concept générique. La spécialisation consiste à supprimer d'abord les propriétés du concept spécifique qui sont communes aux deux concepts (a et b) et à ajouter un lien de spécialisation entre les deux concepts (c).

Exemple

Soient deux composants dont le premier comporte le concept *"Acteur"* et le deuxième le concept *"Agent"*. Supposons que le premier composant permette d'exprimer les interactions entre l'utilisateur du système et le système lui-même. L'*acteur* dans ce composant définit un utilisateur externe du système. Par contre, le deuxième composant est capable d'exprimer les interactions internes au système et le concept *agent* peut représenter non seulement un utilisateur externe mais aussi un objet interne du système. Par conséquent, on peut décider que le concept *"Acteur"* représente une spécialisation du concept *"Agent"*. Ainsi, l'assemblage des deux composants est possible par la spécialisation du concept *"Agent"* en concept *"Acteur"* en appliquant l'opérateur `SPECIALISER`. Ceci est illustré à la Figure 99.

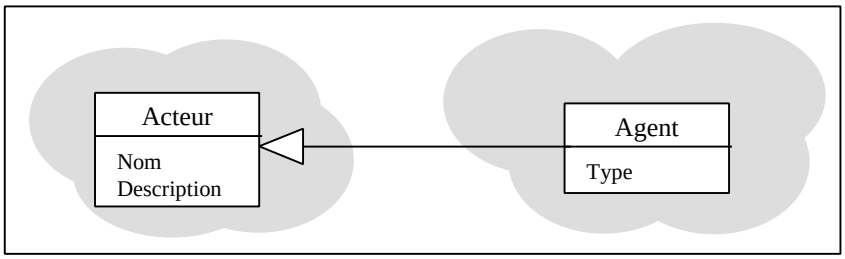


Figure 99 : Exemple d'application de l'opérateur `SPECIALISER`

1.1.2.1.5 GENERALISER

Motivation

Cet opérateur traite le cas où les deux modèles de produit à assembler contiennent deux concepts qui ont une sémantique similaire, mais sont différents du point de vue de leur structure. Afin de permettre la cohabitation des deux concepts dans le même modèle, il est possible de les généraliser dans un concept générique qui assure la connexion entre les modèles de produit initiaux.

Formalisme

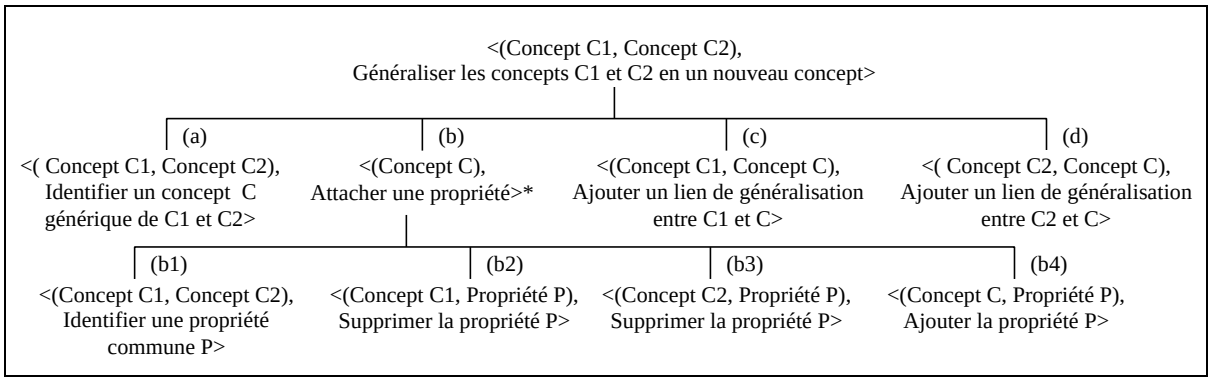
`GENERALISER : C * C → C`

GENERALISER(c_{1i},c_{2j}) = $c_k \cup l_{ik}$: est un(c_{1i},c_k) $\cup$ l_{jk}: est un(c_{2j},c_k)| $c_{1i} \in mp_1$; $c_{2j} \in mp_2$; $c_k \in mp_{as}$; $l_{ik} \in mp_{as}$; $l_{kj} \in mp_{as}$

Démarche

Signature : <(Concept C1, Concept C2), Généraliser les concepts C1 et C2 en nouveau concept>

Corps :



Description

L'opérateur GENERALISER permet de généraliser deux ou plusieurs concepts en un nouveau concept. Le concept générique est créé (a) et relié aux concepts spécifiques par des liens de généralisation (c et d). Les propriétés communes des concepts spécifiques sont supprimées (b1, b2 et b3) et associées au concept générique (b4).

Exemple

La Figure 100 montre un exemple d'assemblage de deux composants incluant tous les deux la notion de but. Dans le premier composant le but a une structure bien définie, composée d'un *"Verbe"*, d'une *"Cible"*, d'une *"Manière"* etc., tandis que dans le deuxième composant un but est défini de manière informelle à l'aide de la propriété *"Description"*. La généralisation des deux concepts en un nouveau concept *"But"* permet de faire le lien entre les deux composants.

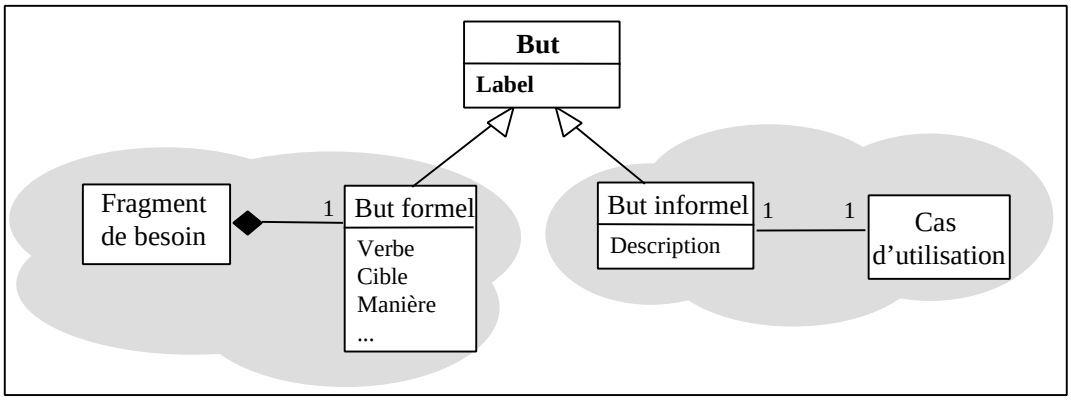


Figure 100 : Exemple d'application de l'opérateur GENERALISER

1.1.2.2 Opérateurs de suppression

Dans cette section nous présentons les opérateurs de suppression qui permettent d'éliminer certains éléments du modèle de produit en construction.

1.1.2.2.1 SUPPRIMER_CONCEPT

Motivation

Lors de l'assemblage des parties de produit de deux composants, un concept peut devenir inutile pour des raisons différentes :

- un nouveau concept ajouté dans le modèle de produit en construction le remplace,
- il existe un autre concept ayant la même sémantique mais de structure plus riche qui peut le remplacer,
- dans le modèle de produit en construction ce concept existe en double,
- l'assemblage des modèles de processus correspondants a éliminé les directives le manipulant.

L'opérateur `SUPPRIMER_CONCEPT` permet de supprimer un concept du modèle de produit. Il ne peut être appliqué que sur le modèle de produit en cours de construction.

Formalisme

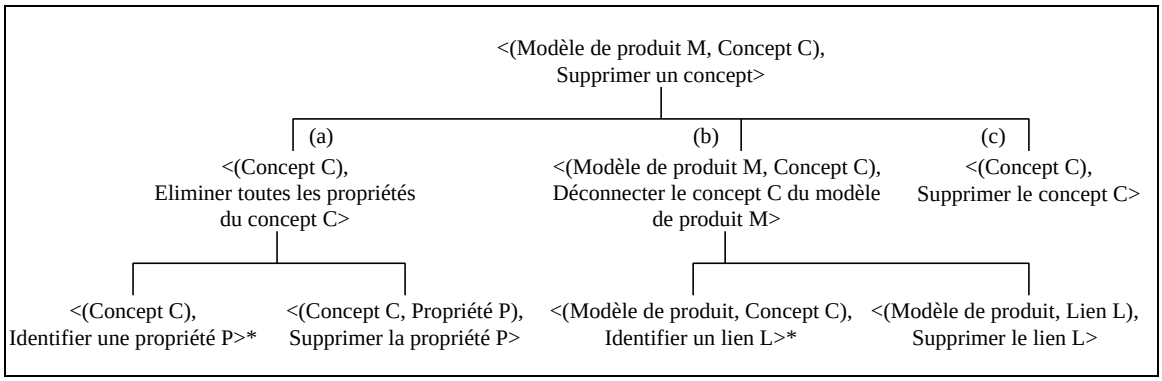
$\texttt{SUPPRIMER\_CONCEPT} : MP * C \rightarrow MP$

$\texttt{SUPPRIMER\_CONCEPT}(mp, c_i) = mp \setminus c_i$

Démarche

Signature : <(Modèle de produit, Concept C), Supprimer un concept>

Corps :



Description

La suppression d'un concept inclut la suppression de toutes ses propriétés (a) et de tous les liens (b) qui le relient au reste du modèle.

Exemple

La Figure 101 montre le résultat intermédiaire de l'assemblage de deux modèles de produit. Le modèle en cours de construction contient deux concepts définissant deux types de scénario *"Scénario semi-formel"* et *"Scénario informel"*. L'ingénieur d'applications a deux possibilités : soit garder les deux types de scénarios et généraliser les concepts correspondants en un nouveau concept *"Scénario"*, soit garder un des deux concepts et supprimer l'autre. Supposons, que l'ingénieur d'applications décide de garder uniquement les scénarios semi-formels. L'opérateur `SUPPRIMER_CONCEPT` permet d'éliminer le concept *"Scénario informel"* du modèle final en éliminant d'abord toutes les propriétés de ce concept et tous les liens qui l'associent au reste du modèle. Les opérateurs `SUPPRIMER_PROPRIETE` et `SUPPRIMER_LIEN` sont appliqués respectivement. Les deux opérateurs en question sont présentés plus loin dans ce chapitre.

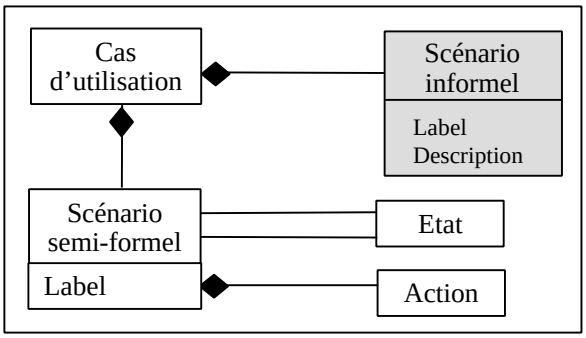


Figure 101 : Exemple d'application d'opérateur `SUPPRIMER-CONCEPT`

1.1.2.2.2 SUPPRIMER_LIEN

Motivation

L'opérateur `SUPPRIMER_LIEN` permet de supprimer un lien entre deux concepts du modèle de produit. Cet opérateur permet de simplifier le modèle de produit assemblé.

Formalisme

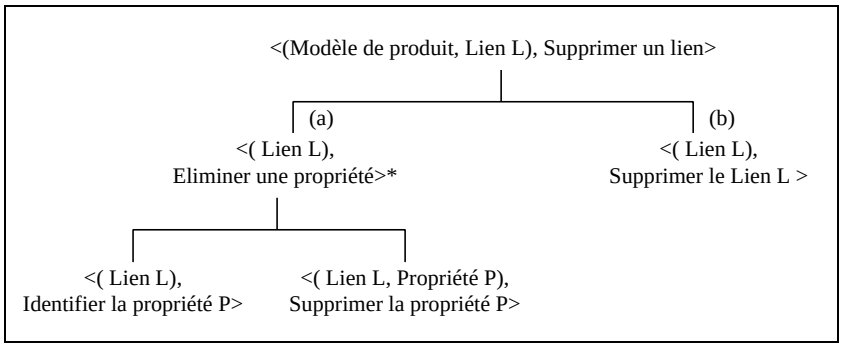
$\texttt{SUPPRIMER\_LIEN} : MP * L \rightarrow MP$

$\texttt{SUPPRIMER\_LIEN}(mp, l_{ij}) = mp \setminus l_{ij}(c_i, c_j) \mid c_i, c_j \in mp$

Démarche

Signature : <(Modèle de produit, Lien L), Supprimer un lien>

Corps :



Description

La suppression d'un lien entre deux concepts dans un modèle de produit conduit à supprimer toutes les propriétés structurelles du lien (a) d'abord et ensuite à supprimer le lien lui-même (b). Afin de pouvoir appliquer cet opérateur, il faut vérifier qu'aucun des deux concepts impliqués dans le lien ne soit déconnecté du modèle de produit. Si l'un des concepts n'a pas d'autres liens avec le reste du modèle, soit le lien ne peut pas être supprimé, soit le concept doit aussi être supprimé.

Exemple

Dans l'exemple précédent, la suppression d'un concept implique la suppression de tous les liens qui le connectent au reste du modèle de produit en construction. Pour pouvoir supprimer le concept *"Scénario informel"* (Figure 101) on doit supprimer d'abord tous les liens le reliant au reste du modèle de produit.

1.1.2.2.3 SUPPRIMER_PROPRIETE

Motivation

L'opérateur `SUPPRIMER_PROPRIETE` permet de supprimer une propriété p_{ik} d'un concept c_i ou d'un lien l_i. Il est appliqué souvent simultanément aux opérateurs `SUPPRIMER_CONCEPT` et `SUPPRIMER_LIEN`. Il peut aussi être utile après la fusion de deux concepts de modèles de produit différents en un nouveau concept dans le modèle en cours de construction lorsque certaines propriétés sont en double (voir la définition et l'exemple de l'opérateur `FUSIONNER_CONCEPT`).

Formalisme

$$SUPPRIMER\_PROPRIETE : C * P \rightarrow C$$

$$SUPPRIMER\_PROPRIETE(c_i, p_{ik}) = c_i \setminus p_{ik}(c_i) \mid c_i \in mp$$

ou

$$SUPPRIMER\_PROPRIETE : L * P \rightarrow L$$

$$SUPPRIMER\_PROPRIETE(l_{ij}, p_{ijk}) = l_{ij} \setminus p_{ijk}(l_{ij}) \mid l_{ij} \in mp$$

Démarche

Signature : <(Concept C, Propriété P), Supprimer la propriété P>

Corps : Actions :

1. Supprimer la propriété du concept

ou

Signature : <(Lien L, Propriété P), Supprimer la propriété P>

Corps : Actions :

1. Supprimer la propriété du lien

Description

La démarche de cet opérateur est présentée par une directive exécutable composée d'une action, qui supprime la propriété du concept si c'est une propriété de concept ou la propriété du lien si c'est une propriété de lien.

Exemple

Dans l'exemple précédent (Figure 101), la suppression du concept *"Scénario informel"* implique la suppression de toutes ses propriétés : *"Label"*, *"Description"*.

1.1.2.3 Opérateurs d'unification

L'assemblage de deux éléments peut nécessiter de les renommer d'abord. Les opérateurs d'unification permettent de changer les noms des concepts, des liens ou des propriétés attachées aux concepts et aux liens. En effet les éléments des différents modèles de produit peuvent avoir des sémantiques similaires mais être nommés différemment. Afin de pouvoir les assembler correctement dans le modèle de produit en cours de construction il faut qu'ils aient le même nom. Par contre, les éléments ayant le même nom dans les deux modèles de produit mais des sémantiques ou des structures différentes doivent être renommés avant d'être assemblés.

Les trois opérateurs décrits ci-dessous sont proposés pour renommer respectivement les concepts, les liens et les propriétés.

1.1.2.3.1 RENOMMER_CONCEPT

Motivation

L'opérateur `RENOMMER_CONCEPT` permet de changer le nom d'un concept. Les concepts ayant le même nom dans deux modèles de produit à assembler doivent être similaires du point de vue sémantique et du point de vue structurel sinon il faut en renommer au moins un. De la même manière, quand deux concepts de modèles différents ont la même sémantique et la même structure, le processus d'assemblage des modèles de produit nécessite qu'ils aient aussi le même nom. Ainsi, au moins un des

deux concepts doit être renommé. L'opérateur RENOMMER_CONCEPT est surtout utile quand deux modèles de produit se recouvrent, c'est à dire quand les deux méthodes sont destinées à construire des produits similaires.

Formalisme

$RENOMMER\_CONCEPT : C \rightarrow C$

$RENOMMER\_CONCEPT(c_i) = c_i:nouveau\_nom\_concept | c_i \in mp$

Démarche

Signature : <(Concept C), Renommer le concept C>

Corps :

Action : Remplacer le nom du concept C par le nouveau nom

Description

Renommer un concept consiste à changer son nom. Le nouveau nom doit être unique dans le modèle de produit. Puisque tout concept du modèle intégré doit avoir un nom unique, s'il existe des concepts ayant des noms identiques il faut vérifier si leur sémantique et leur structure sont similaires. Si c'est le cas, les concepts doivent être fusionnés sinon l'un des deux doit être renommé.

Exemple

Le concept *"Scénario"* existe dans le composant CREWS-*L'Ecritoire* et dans le composant représentant le modèle des cas d'utilisation de la méthode OOSE. La sémantique de ce concept est similaire dans les deux composants mais leur structure est différente. Pour pouvoir intégrer ces deux composants il faut renommer le concept *"Scénario"* au moins dans un des composants. Puisque le *"Scénario"* de CREWS-*L'Ecritoire* est plus structuré et plus formalisé que celui de OOSE, on peut choisir de les renommer tous les deux en *"Scénario formel"* et *"Scénario informel"* respectivement.

1.1.2.3.2 RENOMMER_LIEN

Motivation

L'opérateur RENOMMER_LIEN est utilisé pour valider le modèle de produit intégré. Il permet de changer le nom d'un lien.

Si après assemblage de deux modèles de produit, deux concepts sont reliés par deux liens ayant le même nom, alors un de ces liens doit être renommé ou supprimé du modèle de produit intégré ou bien les deux liens doivent être fusionnés (section 1.1.2.5).

Formalisme

RENOMMER_LIEN : L → L

RENOMMER_LIEN(l_{ij}) = l_{ij}:nouveau_nom_lien| $l_{ij} \in$ mp

Démarche

Signature : <(Lien L), Renommer le lien L>

Corps :

Action : Changer le nom du lien L par le nouveau nom

Description

Renommer un lien consiste à changer son nom. Tout lien entre deux concepts d'un modèle de produit doit avoir un nom unique. S'il existe des liens entre deux concepts ayant des noms identiques il faut vérifier si leur sémantique et leur structure sont similaires. Si c'est le cas, les liens doivent être fusionnés, l'un des deux liens doit être supprimé ou l'un des deux liens doit être renommé.

1.1.2.3.3 RENOMMER_PROPRIETE

Motivation

L'opérateur RENOMMER_PROPRIETE peut être utilisé pour valider le modèle de produit assemblé. Il permet de changer le nom d'une propriété associée à un concept ou à un lien.

L'opérateur RENOMMER_PROPRIETE peut être utile dans le cas de fusion de deux concepts de même nom lorsque le concept fusionné préserve toutes les propriétés des deux concepts initiaux (voir les opérateurs de fusion dans la section 1.1.2.5). Certaines propriétés peuvent être obtenues en double. Si la sémantique et/ou la structure des propriétés en double est différente, une des deux propriétés doit être renommée, sinon une des deux propriétés doit être supprimée.

Formalisme

RENOMMER_PROPRIETE : C * P → P

RENOMMER_PROPRIETE(c_i,p_{ik}) = p_{ik}:nouveau_nom_propriété(c_i), $c_i \in$ mp

ou

RENOMMER_PROPRIETE : L * P → P

RENOMMER_PROPRIETE(l_{ij},p_{ijk}) = p_{ijk}:nouveau_nom_propriété(l_{ij}), $l_i \in$ mp

Démarche

Signature : <(Concept C, Propriété P), Renommer la propriété P>

ou

Signature : <(Lien L, Propriété P), Renommer la propriété P>

Corps :

Action : Changer le nom de la propriété P par le nouveau nom

Description

Renommer une propriété consiste à changer son nom. Toutes les propriétés attachées à un concept ou à un lien doivent avoir des noms différents. Si lors de l'assemblage de modèles de produit un nouveau concept (ou lien) est obtenu et qu'il a deux propriétés avec des noms identiques, alors une des propriétés doit être renommée ou l'une doit être supprimée.

Exemple

La nécessité de renommer une propriété structurelle peut être la conséquence du fait que l'on a renommé un concept ou un lien précédemment. Supposons que nous ayons renommé le concept "Agent" en "Acteur", en conséquence, la propriété "Nom_Agent" doit être renommée "Nom_Acteur".

1.1.2.4 Opérateurs de modification

Les opérateurs de modification sont utilisés pour unifier les modèles de produit avant de procéder à leur assemblage. Les opérateurs proposés dans cette section permettent de transformer un lien ou une propriété en un nouveau concept.

1.1.2.4.1 OBJECTIFIER_LIEN

Motivation

L'opérateur `OBJECTIFIER_LIEN` permet de transformer un lien entre deux concepts $l_{ij}(c_i, c_j)$ en un nouveau concept c_k et deux nouveaux liens $l_{ik}(c_i, c_k)$ et $l_{kj}(c_k, c_j)$ connectant le nouveau concept avec les deux concepts initiaux. Cet opérateur peut être utile dans l'unification des deux modèles de produit préalable à leur assemblage.

Formalisme

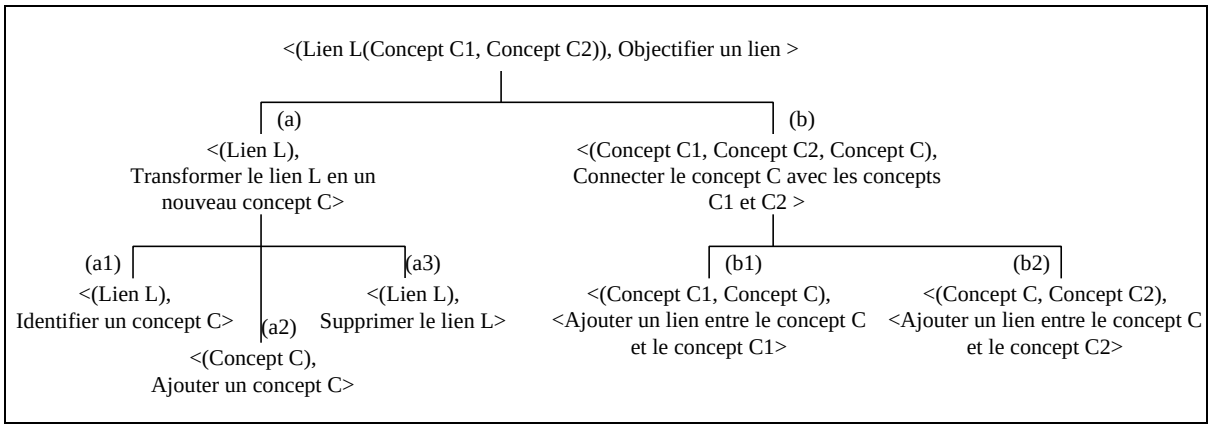
$$OBJECTIFIER\_LIEN : C * C * L \rightarrow C * L * L$$

$$OBJECTIFIER\_LIEN(c_i, c_j, l_{ij}(c_i, c_j)) = mp \setminus l_{ij}(c_i, c_j) \cup c_k \cup l_{ik}(c_i, c_k) \cup l_{kj}(c_k, c_j) \mid c_i, c_j \in mp$$

Démarche

Signature : <(Lien L (Concept C1, Concept C2)), Objectifier un lien >

Corps :



Description

Objectifier un lien consiste à transformer un lien entre deux concepts en un nouveau concept (a). Pour cela il faut créer un nouveau concept en appliquant l'opérateur `AJOUTER_CONCEPT` (a2) et supprimer le lien avec l'opérateur `SUPPRIMER_LIEN` (a3). Puis le nouveau concept doit être connecté aux concepts initiaux en appliquant l'opérateur `AJOUTER_LIEN` (b1 et b2).

Exemple

Sur la Figure 102, le lien *"OU"* entre les *"Fragments de besoin"* dans le composant de méthode CREWS-*L'Ecritoire* relie les scénarios qui décrivent les buts alternatifs les uns aux autres. L'ensemble de tels scénarios représente un cas d'utilisation. Pour faire apparaître explicitement la notion de cas d'utilisation on peut objectifier le lien *"OU"* entre les *"Fragments de besoin"* et le transformer en un nouveau concept appelé *"Cas d'utilisation"* connecté au *"Fragment de besoin"*.

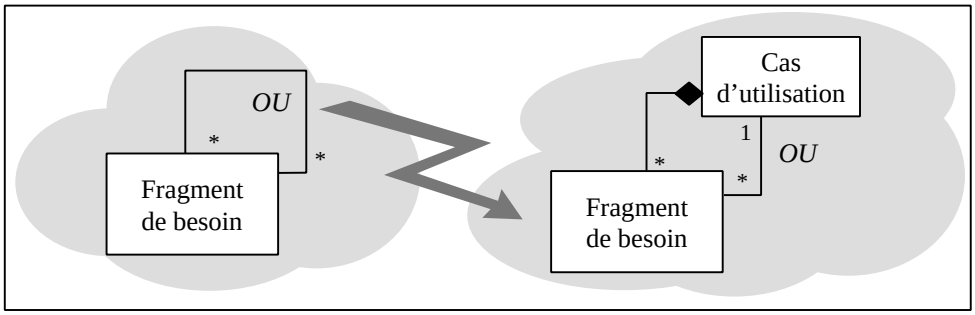


Figure 102 : Exemple d'application de l'opérateur `OBJECTIFIER_LIEN`

1.1.2.4.2 OBJECTIFIER_PROPRIETE

Motivation

L'opérateur `OBJECTIFIER_PROPRIETE` permet de transformer une propriété d'un concept en un nouveau concept. Cet opérateur peut aussi être utilisé pour l'unification des deux modèles de produit avant leur intégration.

Formalisme

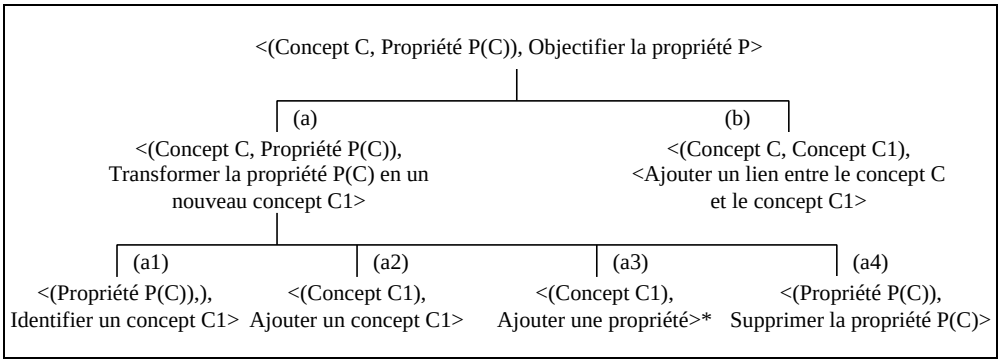
$OBJECTIFIER\_PROPRIETE : C * P \rightarrow MP$

$OBJECTIFIER\_PROPRIETE(c_i, p_{ik}) = c_i \setminus p_{ik}(c_i) \cup c_k \cup l_{ik}(c_i, c_k) \mid c_i \in mp$

Démarche

Situation : <(Concept C, Propriété P(C)), Objectifier la propriété P>

Corps :



Description

Transformer une propriété en un concept (a) consiste à créer un nouveau concept (a1, a2, a3), à supprimer l'ancienne propriété (a4) et à connecter le nouveau concept au concept initial (b).

Exemple

Par exemple, dans le composant de méthode définissant le modèle des cas d'utilisation de la méthode OOSE le concept *"Cas d'utilisation"* a une propriété appelée *"Objectif"* qui exprime le but devant être accompli par le système et que le cas d'utilisation décrit. Pour faciliter l'assemblage de ce composant avec le composant CREWS-*L'Ecritoire* dans lequel la notion de but du système est exprimée explicitement par un concept *"But"* nous proposons de transformer la propriété *"Objectif"* en un concept *"Objectif"* dans le composant de la méthode OOSE. Ceci permet de comparer ce concept avec le concept correspondant de CREWS-*L'Ecritoire*. La Figure 103 montre la transformation du modèle de produit effectuée à l'aide de l'opérateur `OBJECTIFIER_PROPRIETE`.

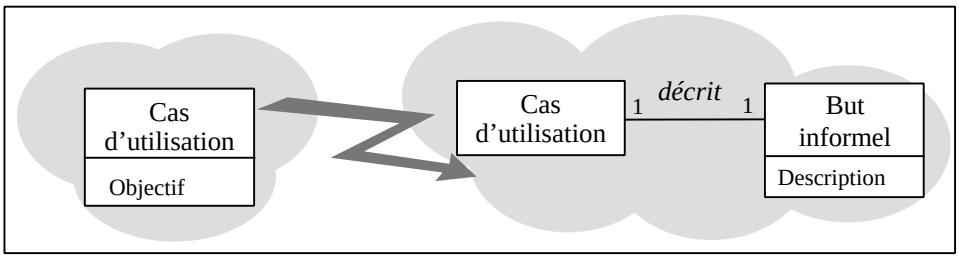


Figure 103 : Exemple d'application de l'opérateur `OBJECTIFIER_PROPRIETE`

1.1.2.5 Opérateurs de fusion

Les opérateurs de fusion permettent d'assembler les modèles de produit ayant des éléments communs sans ajout de nouveaux concepts ou de nouveaux liens.

1.1.2.5.1 FUSIONNER_CONCEPT

Motivation

L'opérateur `FUSIONNER_CONCEPT` permet de fusionner deux concepts d'origines différentes en un nouveau concept dans le modèle de produit en cours de construction.

Formalisme

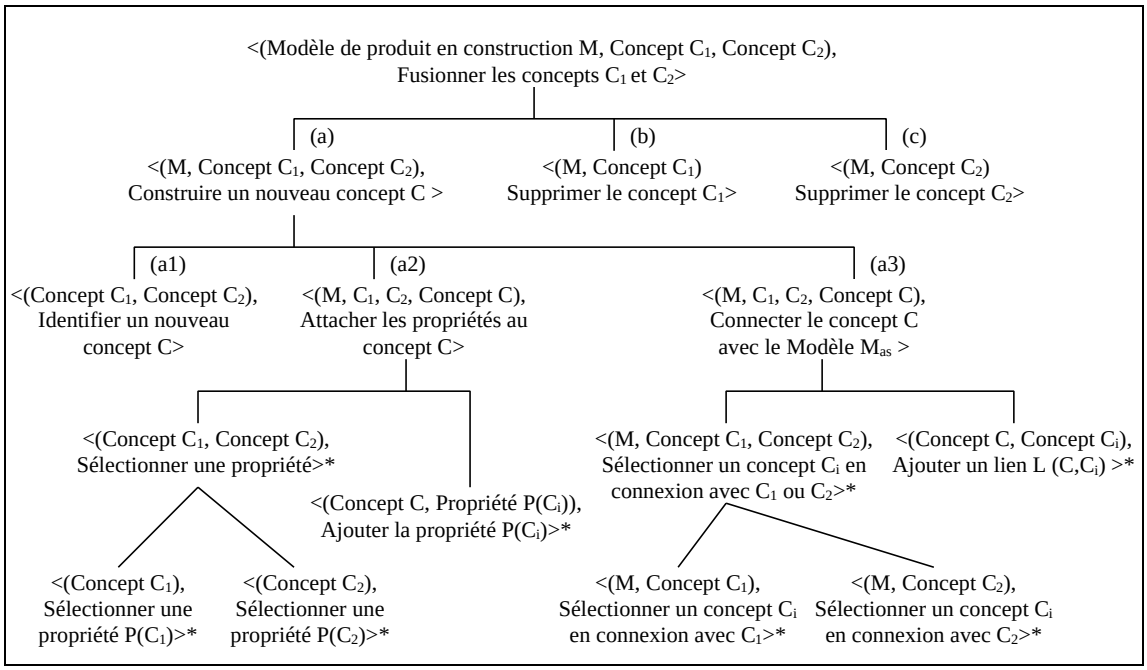
$\texttt{FUSIONNER\_CONCEPT} : M^* \; C \; * \; C \rightarrow C$

$\texttt{FUSIONNER\_CONCEPT}(mp_{as}, c_1, c_2) = c_3 \mid c_1 \in mp_1 \; ; \; c_2 \in mp_2 \; ; \; c_3 \in mp_{as} \; ;$

Démarche

Situation : <(Modèle de produit en construction M_{as}, Concept C_1, Concept C_2), Fusionner les concepts C_1 et C_2>

Corps :



Description

La fusion de deux concepts est possible si leurs noms sont identiques, si leur sémantique est la même et si leurs structures sont similaires. Les mesures de similarité sémantique et de similarité structurelle sont présentées à la section 2.1.

L’opérateur FUSIONNER_CONCEPT supprime les deux concepts initiaux (b et c) et crée un nouveau concept (a) qui conserve les propriétés des deux concepts initiaux (a2) ainsi que leurs liens avec d’autres concepts (a3).

Dans le cas où les concepts ont certaines différences structurelles, l’ingénieur de méthodes doit choisir entre la généralisation des deux concepts et leur fusion. La généralisation permet de faire cohabiter les deux concepts dans le nouveau modèle de produit tandis que la fusion crée un nouveau concept à partir des deux concepts initiaux en supprimant les concepts de départ. Le choix de l’opérateur dépend de la solution optée pour l'assemblage des modèles des processus correspondants. Si le modèle de processus intégré a gardé des directives manipulant les deux types de concept, l’ingénieur de méthodes doit appliquer l’opérateur de généralisation et garder les deux concepts dans le modèle de produit intégré, sinon il doit modifier le modèle de processus intégré. Vice versa, quand l'assemblage des modèles de produit se fait avant l'assemblage des modèles de processus il faut savoir que le choix d’opérateur va impliquer le choix de la démarche intégrée.

La fusion des deux concepts génère un nouveau concept qui peut avoir certaines propriétés en double, c’est à dire ayant les mêmes noms. Si les deux propriétés de même nom ont les mêmes domaines, une des deux propriétés doit être supprimée sinon une des deux doit être renommée ou bien elles peuvent être fusionnées. Si les propriétés ont des noms différents qui représentent des synonymes, deux solutions sont possibles : soit une des deux propriétés est supprimée en gardant celle qui est la plus appropriée, soit elles sont fusionnées mais l'une est nécessairement renommée.

De la même manière, les liens connectant le nouveau concept avec le reste du modèle doivent être validés car il peut arriver qu’après la fusion le nouveau concept soit relié à un autre concept du modèle intégré par deux liens qui ont le même nom. Si les deux liens ont la même sémantique, un parmi eux doit être supprimé, sinon un des deux liens doit être renommé ou bien les deux liens doivent être fusionnés.

Exemple

Le concept de *“Cas d'utilisation”* est défini dans le composant de OOSE et dans le composant de CREWS-*L’Ecritoire* (après l'application de l'opérateur OBJECTIFIER_LIEN à la section 1.1.2.4.1). Afin de pouvoir fusionner ces deux concepts on doit mesurer d'abord leur similarité sémantique ainsi que leur similarité structurelle. Les deux concepts ont la même sémantique, ils représentent tous les deux les fonctionnalités que le système doit offrir à ses utilisateurs. Les structures des deux concepts sont similaires, chacun d'eux est composé d'un ensemble de scénarios. Par conséquent, nous pouvons fusionner les deux concepts *“Cas d'utilisation”*. Pour plus de détail sur les mesures de similarité des éléments de modèles de produit voir la section 2.1.

1.1.2.5.2 FUSIONNER_LIEN

Motivation

Cet opérateur permet de fusionner deux liens d’un modèle de produit en un nouveau lien. Les deux liens initiaux sont supprimés du modèle. L’opérateur FUSIONNER_LIEN ne peut être appliqué que

sur le modèle de produit en cours de construction. De plus, il ne peut être utilisé qu'après l'application de l'opérateur FUSIONNER_CONCEPT sur les concepts impliqués dans les liens en question.

Formalisme

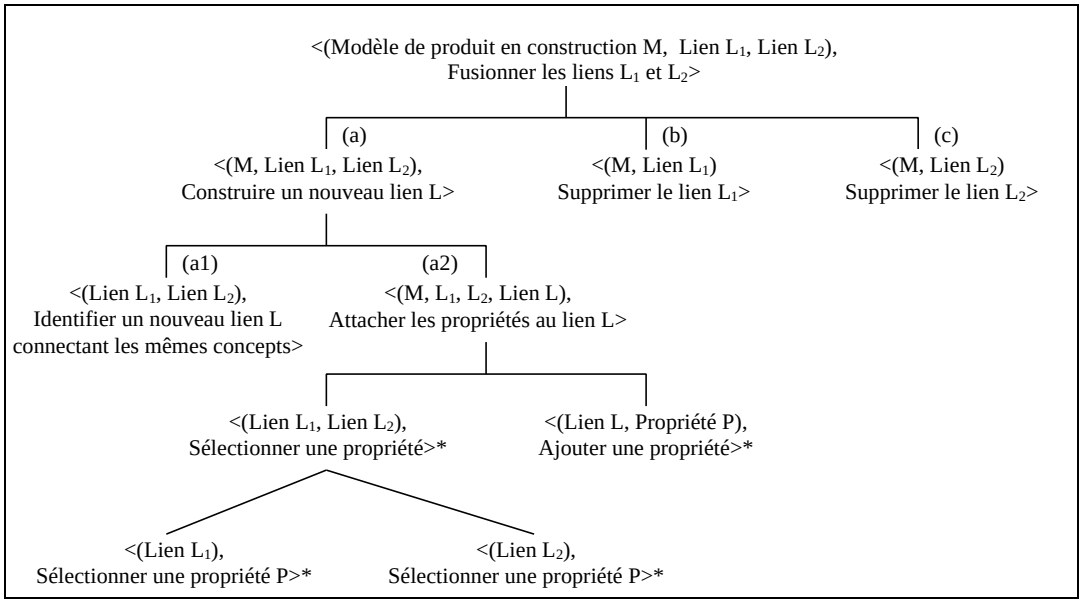
$FUSIONNER\_LIEN : MP * L * L \rightarrow L$

$FUSIONNER\_LIEN(mp, l_1, l_2) = mp_3 \cup l_{12} \setminus l_1 \cup l_2$

Démarche

Signature : <(Modèle de produit en construction M_{as}, Lien L_1, Lien L_2), Fusionner les liens L_1 et L_2>

Corps :



Description

S'il existe deux liens similaires (ayant les mêmes noms et les mêmes propriétés structurelles) connectant les mêmes concepts, un des deux liens doit être supprimé. S'il existe des différences entre leurs propriétés structurelles, les liens peuvent être fusionnés ou bien un des deux liens peut être éliminé en sachant que cela implique des modifications sur le modèle de processus assemblé.

La fusion de deux liens consiste à créer un nouveau lien ayant le même nom (a) et toutes les propriétés des deux liens, puis à supprimer les deux liens initiaux (b, c).

Exemple

1.1.2.5.3 FUSIONNER_PROPRIETE

Motivation

L'opérateur FUSIONNER_PROPRIETE permet de fusionner deux propriétés d'un concept ou d'un lien. Il ne peut être appliqué que sur le modèle de produit intégré et seulement après l'application de l'opérateur FUSIONNER_CONCEPT et / ou l'opérateur FUSIONNER_LIEN.

Formalisme

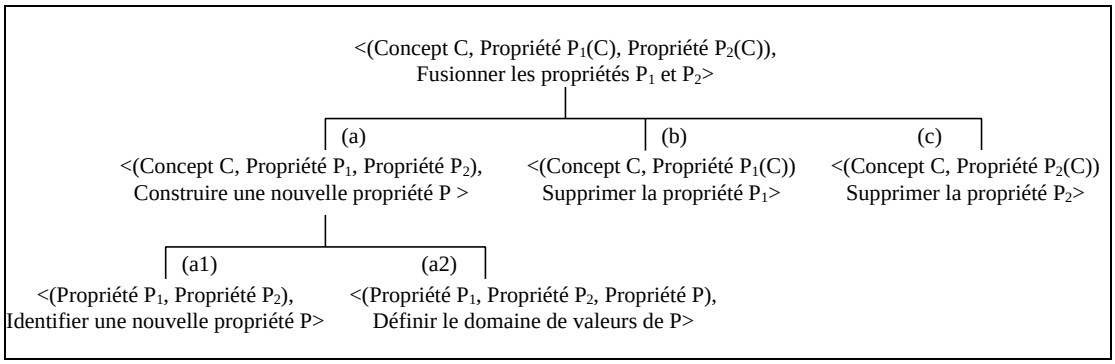
$FUSIONNER\_PROPRIETE : C * P * P \rightarrow P$

$FUSIONNER\_PROPRIETE(c_i, p_{i1}, p_{i2}) = c_i \cup p_{i12}(c_i) \setminus p_{i1}(c_i), p_{i2}(c_i) \mid c_i \in mp$

Démarche

Signature : <(Concept C, Propriété P_1(C), Propriété P_2(C)), Fusionner les propriétés P_1 et P_2>

Corps :



Description

Si un concept ou un lien a deux propriétés similaires (les mêmes noms, les mêmes domaines), une des deux propriétés doit être supprimée. Si leurs domaines sont différents, elles peuvent être fusionnées en une nouvelle propriété en choisissant le domaine le plus approprié. Ainsi une nouvelle propriété est créée (a) et les propriétés initiales sont supprimées (b, c).

1.2 Opérateurs d'assemblage des modèles de processus

Dans cette section, nous présentons les opérateurs permettant l'intégration des modèles de processus. Nous représentons les modèles de processus des méthodes par des cartes de processus et les directives associées. Par conséquent, l'intégration des modèles de processus consiste à intégrer les cartes de processus et à adapter les directives correspondantes.

1.2.1 Méta-concepts de processus

Nous définissons d'abord les méta-concepts utilisés pour définir les opérateurs:

- *I* est un ensemble d'intentions i_k, permettant de définir les objectifs à atteindre dans un processus de développement. Une intention est définie par un label unique i_k et elle a un nom qui est exprimé par un verbe et une cible. Une intention peut être désignée par son label.

```
<intention> ::= <label_intention> : <nom_intention>
<label_intention> ::= un label ordonné, commençant par i et suivi
d'un nombre en index
<nom_intention> ::= <verbe><cible>
<verbe> ::= un verbe qui exprime une intention appartenant à la
démarche modélisée
<cible> ::= <partie_de_produit>
```

- *S* est un ensemble de stratégies s_k permettant de définir les différentes manières de satisfaire les intentions. Une stratégie est définie par un label unique et elle a un nom. Elle peut être désignée par son label.

```
<stratégie> ::= <label_stratégie> : <nom_stratégie>
<label_stratégie> ::= un label ordonné, commençant par s et suivi
d'un nombre en index
<nom_stratégie> ::= un nom de stratégie
```

- *C* est un ensemble de modèles de processus c_k représentés par des cartes où $C \subseteq I*I*S$.

- Chaque triplet $<i_i,i_j,s_{ijk}>$ est appelé section de la carte.

1.2.2 Typologie des opérateurs d'assemblage de modèles de processus

Nous proposons quatre types d'opérateurs pour intégrer les modèles de processus:

1. Les opérateurs d'*addition* permettant d'ajouter des nouveaux éléments dans les cartes de processus.

2. Les opérateurs de *suppression* permettant d'enlever certains éléments des cartes de processus.

3. Les opérateurs d'*unification* permettant de changer les noms des différents éléments dans des cartes de processus.

4. Les opérateurs de *fusion* destinés à fusionner les éléments provenant des deux cartes différentes en un nouvel élément de la carte intégrée.

Nous développons ces opérateurs dans les sous-sections suivantes.

1.2.2.1 Opérateurs d'addition

Les opérateurs d'addition permettent d'ajouter des nouvelles intentions ou des nouvelles stratégies permettant de faire un lien entre les deux cartes lors de l'assemblage des deux cartes de processus. Ils sont surtout utiles dans l'assemblage des cartes de processus qui n'ont aucun élément en commun.

1.2.2.1.1 AJOUTER_INTENTION

Motivation

L'opérateur `AJOUTER_INTENTION` aide à assembler des cartes qui n'ont aucun élément commun. Il permet d'ajouter une nouvelle intention pour faire la connexion entre les cartes.

Formalisme

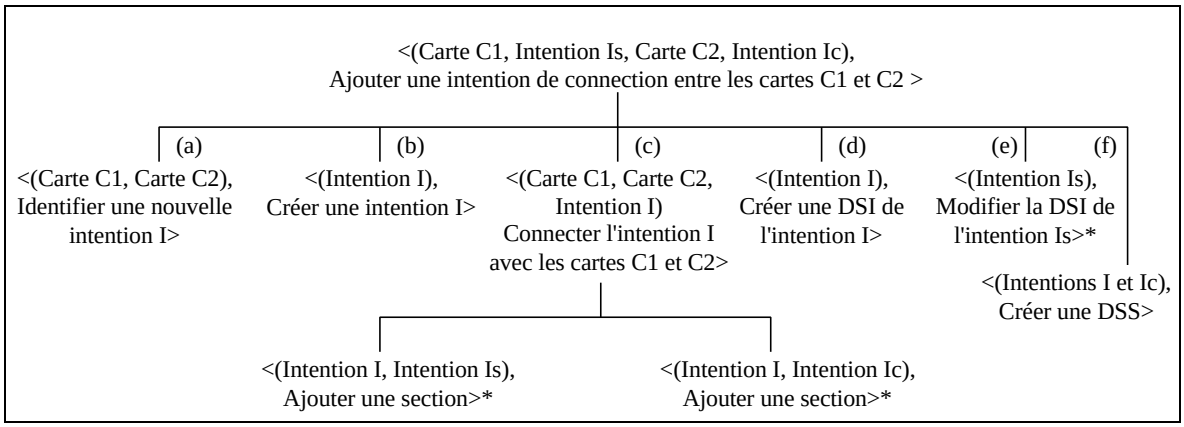
$$AJOUTER\_INTENTION : C * C \rightarrow C$$

$$AJOUTER\_INTENTION(c_1, c_2) = c_1 \cup c_2 \cup i_i \mid i_i \notin c_1 \; ; \; i_i \notin c_2$$

Directive

Signature : <(Carte C1, Carte C2), Ajouter une intention de connexion entre les cartes C1 et C2 >

Corps :



Description

L'opérateur `AJOUTER_INTENTION` permet d'ajouter une nouvelle intention dans une carte de processus lors de l'assemblage de deux cartes. Il ne peut être appliqué que pour connecter deux cartes n'ayant aucune intention en commun. La nouvelle intention sert de connecteur entre les deux cartes et n'a de sens que dans la carte intégrée.

L'ajout d'une nouvelle intention correspond à la création d'une intention (a, b) qui implique la création d'au moins deux sections connectant cette intention avec les deux cartes initiales (c). Il doit y avoir au moins une stratégie permettant d'accéder à la nouvelle intention et une stratégie permettant de sortir de cette intention. Bien sûr, le processus d'assemblage permet d'ajouter autant de sections que nécessaire

pour justifier l'ajout d'une nouvelle intention. L'introduction d'une nouvelle intention devient encore plus intéressante si plusieurs stratégies permettent de satisfaire cette intention et si plusieurs stratégies permettent d'utiliser le résultat obtenu en réalisant l'intention. Une DSI correspondant à la nouvelle intention doit être créée (d). Les DSI des intentions permettant d'accéder à la nouvelle intention doivent être modifiées également (e). Si on ajoute plusieurs sections connectant la nouvelle intention I avec l'intention cible Ic il est nécessaire de créer la DSI correspondant à cette situation.

Exemple

Supposons que l'on veuille assembler deux composants de méthode où le premier propose une démarche pour écrire des scénarios et le deuxième une démarche de construction d'un diagramme de classes. Comme le montre la Figure 104, l'intégration des deux cartes nécessite une nouvelle intention appelée *"Identifier une ressource"* et deux sections connectant cette intention aux deux cartes. Cette nouvelle intention permet de faire le lien entre les deux cartes de processus qui n'avaient aucune intention en commun. Elle permet d'identifier les ressources du système décrites dans les scénarios du premier composant devenant des classes d'objets dans le diagramme de classes du deuxième composant.

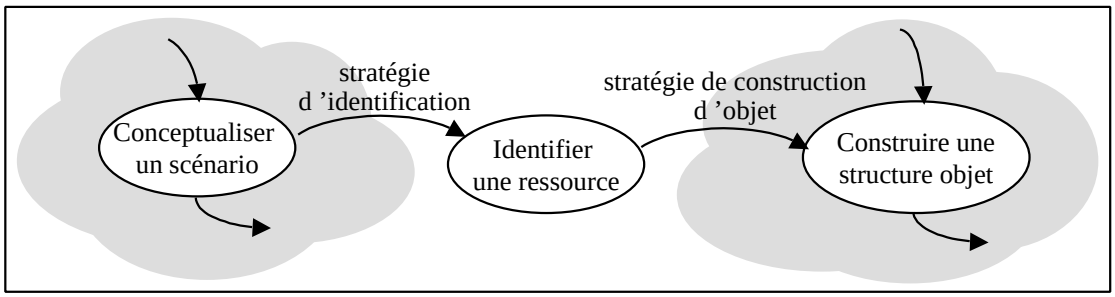


Figure 104 : Exemple d'application de l'opérateur `AJOUTER_INTENTION`

1.2.2.1.2 AJOUTER_SECTION

Motivation

L'opérateur `AJOUTER_SECTION` permet d'assembler deux cartes en ajoutant une nouvelle section. L'intention source de cette section doit appartenir à l'une des deux cartes tandis que l'intention cible doit appartenir à la deuxième carte. La nouvelle section sert à faire la connexion entre les deux cartes. La stratégie de cette section définit la manière de réaliser l'intention cible à partir de l'intention source.

Formalisme

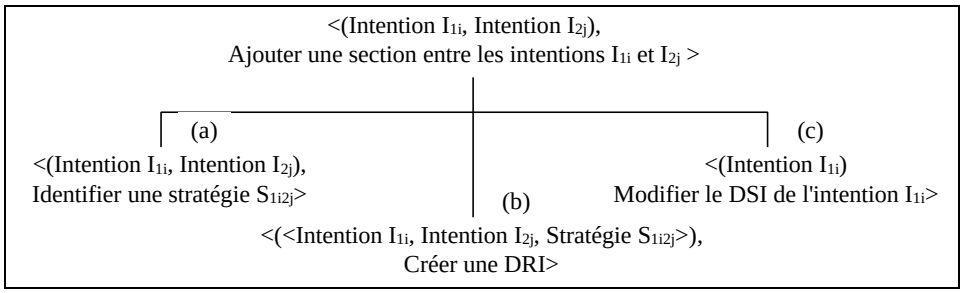
$\texttt{AJOUTER\_SECTION} : I * I \rightarrow I * I * S$

$\texttt{AJOUTER\_SECTION}(i_{1i}, i_{2j}) = \langle i_{1i}, i_{2j}, s_{ijk} \rangle | \; i_{1i} \in c_1 \; ; \; i_{2j} \in c_2$

Directive

Signature : <(Intention I_{1i}, Intention I_{2j}), Ajouter une section entre les intentions I_{1i} et I_{2j}>

Corps :



Description

L'opérateur `AJOUTER_SECTION` est utilisé pour ajouter de nouvelles sections dans la carte assemblée. Cet opérateur permet de faire un lien entre deux cartes qui manipulent des produits différents lorsque le passage d'un produit vers un autre n'est défini dans aucune des cartes. Dans ce cas, un nouveau processus doit être défini par l'ingénieur de méthode. Par conséquent, l'opérateur `AJOUTER_SECTION` demande de définir une nouvelle stratégie entre deux intentions de cartes différentes (a) et ajouter une directive DRI (b) à la nouvelle section. La signature de cette DRI doit avoir la forme suivante : <(Cible de l'intention I_{1i}), I_{2j} avec S_{1i2j}>, elle doit définir une nouvelle manière de satisfaire l'intention cible de la section. La directive DSI de l'intention source doit être modifiée (c) en ajoutant un nouveau choix permettant de sélectionner la nouvelle DRI.

Exemple

Supposons que nous ayons à intégrer deux composants de méthode tel que le premier propose des directives pour conceptualiser un ensemble de scénarios décrivant les services proposés par le système et le deuxième des directives pour représenter ces services en termes de cas d'utilisation (un cas d'utilisation étant un ensemble de scénarios). Les deux composants sont complémentaires. Le premier aide à écrire des scénarios tandis que le deuxième aide à structurer le comportement du système en un ensemble de services. Pour intégrer les deux composants, nous proposons d'ajouter une section entre l'intention *"Conceptualiser un scénario"* du premier composant et l'intention *"Conceptualiser un cas d'utilisation"* du deuxième composant avec la *"stratégie d'intégration"*. La DRI associée à cette section doit aider à regrouper les scénarios en cas d'utilisation.

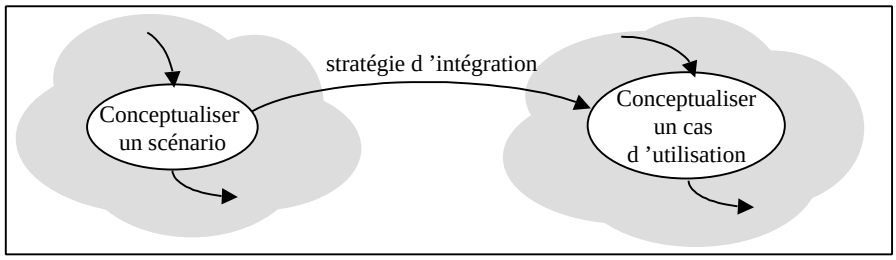


Figure 105 : Exemple d'application de l'opérateur `AJOUTER_SECTION`

1.2.2.2 Opérateurs de suppression

Les opérateurs de suppression permettent d'éliminer certains éléments de la carte de processus. Lors de l'assemblage des cartes, certaines intentions et/ou certaines sections peuvent devenir inutiles pour plusieurs raisons :

- les deux cartes à assembler proposent des sections similaires mais la DRI associée à une des deux sections est plus riche que l'autre ; dans ce cas la section qui a la DRI la moins performante peut être supprimée;
- une des deux cartes propose un processus plus riche et plus détaillé que la directive associée à une section correspondante de la deuxième carte;
- les deux cartes proposent des intentions similaires mais l'intention d'une carte est plus précise que l'intention de l'autre ;
- le produit correspondant à l'intention a été éliminé lors de l'assemblage des modèles de produit;
- etc.

1.2.2.2.1 SUPPRIMER_SECTION

Motivation

L'opérateur `SUPPRIMER_SECTION` permet de supprimer certaines sections de la carte. Il peut être appliqué à une stratégie que l'on considère inutile, ou déjà remplacée par un processus plus pertinent.

Formalisme

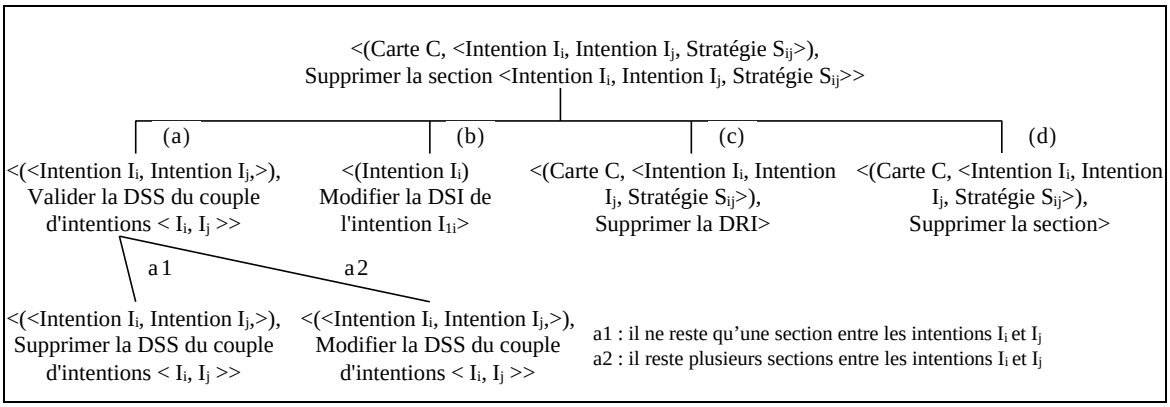
$\texttt{SUPPRIMER\_SECTION} : C * I * I * S \rightarrow C;$

$\texttt{SUPPRIMER\_SECTION}(c,<i_i,i_j,s_{ij}>) = c \setminus <i_i,i_j,s_{ij}>$

Directive

Signature : <(Carte C, <Intention I_i, Intention I_j, Stratégie S_{ij}>), Supprimer la section <Intention I_i, Intention I_j, Stratégie S_{ij}>>

Corps :



Description

L'opérateur `SUPPRIMER_SECTION` peut être appliqué à une section que l'on trouve inutile, insuffisamment intéressante, ou déjà remplacée par un processus plus efficient. La suppression d'une section implique la modification ou la suppression da la directive DSS (a) associée au couple d'intentions concernées, la modification de la directive DSI de l'intention source (b) et la suppression de la directive DRI attachée à la section (c). Si après la suppression de la section, la DSS correspondant au couple d'intentions ne propose qu'un seul choix, il faut supprimer la DSS et modifier la DSI de l'intention source en remplaçant la sélection de DSS par la sélection de la DRI restante. Sinon, modifier la DSS en supprimant le choix de sélectionner la DRI correspondante.

On doit aussi valider la cohérence de la carte en vérifiant qu'après la suppression de la section, l'intention source de la section a d'autres stratégies de sortie et l'intention cible a d'autres stratégies d'entrée. Si une des conditions n'est pas respectée, soit la section ne peut être supprimée soit l'intention violant la condition doit aussi être supprimée de la carte.

Exemple

Supposons que nous avons assemblé précédemment le composant nommé CREWS-*L'Ecritoire* qui permet de découvrir des besoins fonctionnels d'un système sous forme de buts et des scénarios qui les décrivent et le composant issu de la méthode OOSE qui représente les besoins fonctionnels d'un système par des cas d'utilisations. Chaque cas d'utilisation est représenté par un ensemble de scénarios. Les deux composants proposent des directives pour écrire les scénarios. Toutefois, les directives proposées par CREWS-*L'Ecritoir* sont beaucoup plus riches que celle de OOSE. L'ingénieur d'applications peut décider de garder uniquement les directives de CREWS-*L'Ecritoire* et d'éliminer celles de OOSE. Pour cela, il doit appliquer l'opérateur `SUPPRIMER_SECTION` sur la section <*Découvrir un but, Ecrire un scénario, Stratégie libre de* OOSE>. L'application de cet opérateur est illustrée ci-dessous. La stratégie présentée par une flèche pointillée doit être enlevée de la carte finale.

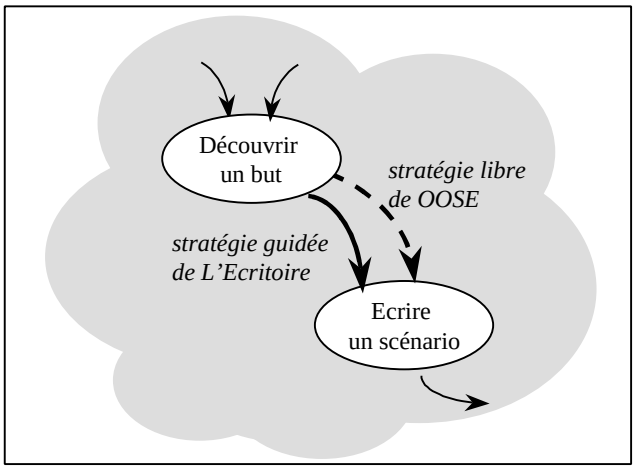


Figure 106 : Exemple d'application de l'opérateur SUPPRIME_SECTION

1.2.2.2.2 SUPPRIMER_INTENTION

Motivation

L'opérateur SUPPRIMER_INTENTION permet d'enlever une intention de la carte et de simplifier ainsi le processus proposé par la carte.

Formalisme

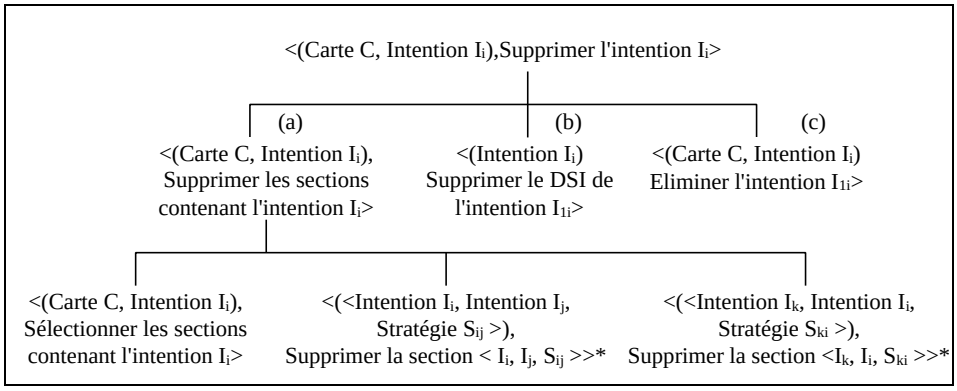
$SUPPRIMER\_INTENTION : C * I \rightarrow C;$

$SUPPRIMER\_INTENTION(c,i_i) = c \setminus \{<i_i,i_j,s_{ijn}> \cup <i_k,i_i,s_{kim}>\} | \forall j,k,m,n$

Directive

Signature : <(Carte C, Intention I_i>), Supprimer l'intention I_i>

Corps :



Description

La suppression d'une intention de la carte requiert de supprimer toutes les sections dans lesquelles l'intention est source ou cible (a). La directive DSI de l'intention doit être également supprimée (b) avant l'élimination de l'intention de la carte (c).

1.2.2.3 Opérateurs d'unification

Les opérateurs d'unification servent à unifier la notation des deux cartes avant de procéder à leur assemblage.

1.2.2.3.1 RENOMMER_INTENTION

Motivation

Le processus d'assemblage conduit parfois à identifier des ressemblances dans les cartes à assembler. Deux intentions de cartes différentes ayant une sémantique similaire doivent avoir la même expression (le même verbe, la même cible) pour que leur assemblage soit possible. On doit choisir l'expression la plus pertinente parmi les deux possibles et renommer la deuxième en sachant que la modification va se propager non seulement sur les directives attachées à la carte mais aussi au modèle de produit correspondant.

L'opérateur `RENOMMER_INTENTION` permet d'unifier la terminologie des cartes à assembler en renommant certaines intentions. Les intentions de cartes différentes ayant des sémantiques similaires doivent avoir les mêmes noms afin de pouvoir assembler les cartes correctement.

Formalisme

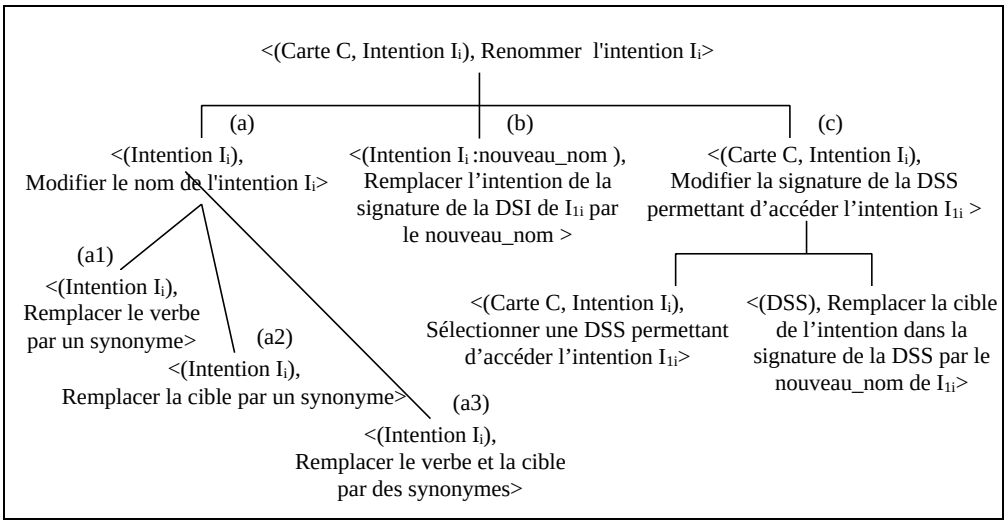
$RENOMMER\_INTENTION : C * I \rightarrow I;$

$RENOMMER\_INTENTION(c, i_i) = c \mid i_i : nouveau\_nom\_intention$

Directive

Signature : <(Carte C, Intention I_i>), Renommer l'intention I_i>

Corps :



Description

L'opérateur `RENOMMER_INTENTION` change le nom d'une intention par un nouveau nom qui est son synonyme. Ce changement consiste à modifier soit le verbe de l'expression (a1), soit la cible (a2) soit les deux (a3). La modification consiste à remplacer le verbe ou/et la cible par des synonymes. La directive DSI associée à l'intention doit être renommée de la même manière (b) ainsi que toutes les DSS qui permettent d'accéder à cette intention (c).

La modification de la cible de l'intention implique des modifications du modèle de produit correspondant, comme par exemple, l'application de l'opérateur `RENOMMER_CONCEPT` sur le concept représentant la cible de l'intention, ou l'application de l'opérateur `RENOMMER_PROPRIETE` si cette cible représente une propriété d'un concept.

Exemple

Supposons que l'on veuille intégrer deux composants de méthode servant à découvrir des besoins fonctionnels du système et dont les démarches sont complémentaires. La carte du premier composant contient l'intention appelée *"Découvrir un but"* tandis que la carte du deuxième composant contient l'intention *"Découvrir un besoin"*. La sémantique des deux intentions est similaire. Afin de pouvoir intégrer les deux intentions (voir l'opérateur `FUSIONNER_INTENTION`) nous devons unifier leurs noms en renommant une des intentions. Par exemple, nous proposons de renommer l'intention *"Découvrir un besoin"* en *"Découvrir un but"* (Figure 107).

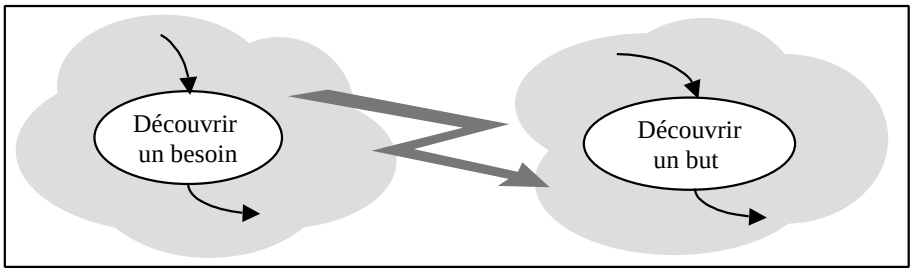


Figure 107 : Exemple d'application de l'opérateur `RENOMMER_INTENTION`

1.2.2.3.2 RENOMMER_SECTION

Motivation

De la même manière que l'opérateur `RENOMMER_INTENTION`, l'opérateur `RENOMMER_SECTION` sert à unifier la terminologie des cartes avant leur assemblage. Les sections connectant les mêmes intentions dans deux cartes différentes et ayant la même sémantique (les mêmes signatures de DRI correspondantes: la même situation et la même intention) mais des noms différents doivent être renommées avant l'assemblage des cartes.

Formalisme

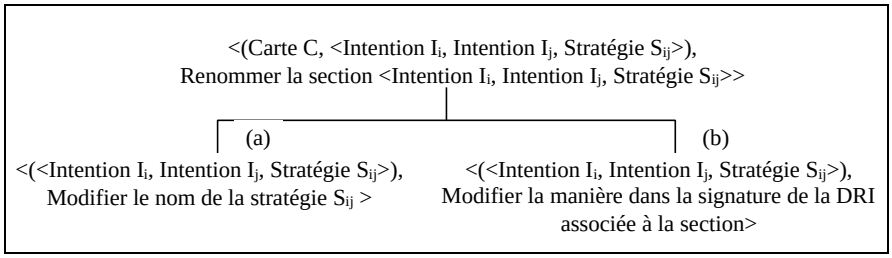
`RENOMMER_SECTION: C * I* I* S → C`

`RENOMMER_SECTION (c,<`i_i`,`i_j`,`s_{ij}`>) = c |<`i_i`,`i_j`,`s_{ij}`:nouveau_nom_stratégie>`

Directive

Signature : <(Carte C, <Intention I_i, Intention I_j, Stratégie S_{ij}>), Renommer la section <Intention I_i, Intention I_j, Stratégie S_{ij}>>

Corps :



Description

L'opérateur `RENOMMER_SECTION` modifie le nom de la stratégie d'une section (a). Cette modification implique des changements correspondants dans la DRI (b) attachée à la section de la carte à laquelle appartient la stratégie. Plus exactement, c'est la manière, dans la signature de la DRI, qui doit être renommée.

Exemple

Supposons que l'on veuille renommer la stratégie de la section *<i_i : Conceptualiser un scénario, i_j : Découvrir un but, s_{ij} : Stratégie alternative>* en *"s_{ij}: Stratégie d'exception"*. La signature de la DRI *"<(Scénario), Découvrir un but avec la stratégie alternative>"* doit aussi être renommée de la même manière *"<(Scénario), Découvrir un but avec la stratégie d'exception>"*.

1.2.2.4 Opérateurs de fusion

Les opérateurs de fusion permettent de fusionner des intentions ou des sections similaires de cartes différentes dans la carte intégrée. Les opérateurs de fusion permettent d'intégrer deux cartes de processus sans ajouter de nouveaux éléments à la carte intégrée.

1.2.2.4.1 FUSIONNER_INTENTION

Motivation

Formalisme

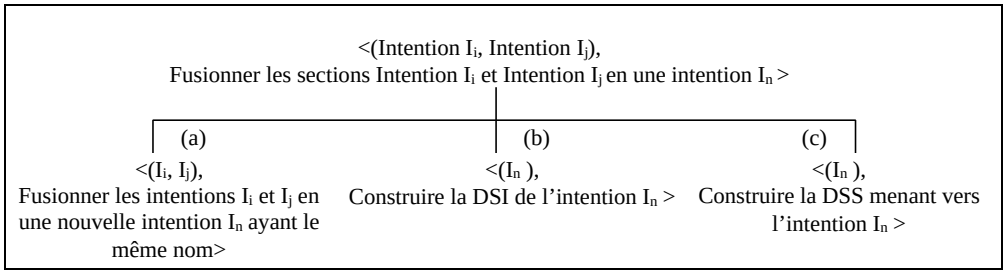
`FUSIONNER_INTENTION : I * I → I;`

`FUSIONNER_INTENTION(`i_1`, `i_2`) = `i_3

Démarche

Signature : <(Intention I_i, Intention I_j), Fusionner les sections Intention I_i et Intention I_j>

Corps :



Description

L'opérateur `FUSIONNER_INTENTION` permet de fusionner deux intentions de cartes différentes ayant des sémantiques similaires.

Les cartes de processus ayant des parties similaires peuvent être intégrées en fusionnant ces parties. Les cartes peuvent avoir des intentions de sémantiques analogues justifiant leur fusion. La fusion de ces intentions permet l'intégration des cartes sans addition de nouvelles intentions ou de nouvelles sections. Pour fusionner ces intentions il faut s'assurer qu'elles aient des noms identiques. L'unification de la terminologie doit être appliquée préalablement à l'assemblage. Si les intentions ont des noms différents, l'opérateur `RENOMMER_INTENTION` doit être appliqué au moins sur une des deux intentions.

La fusion des deux intentions préserve toutes les sections ayant ces intentions comme source ou comme cible. Par conséquent, les DSI des deux intentions sont fusionnées en une nouvelle DSI qui représente un choix entre les DSI initiales (b). Si les intentions fusionnées ont été toutes les deux précédemment accessibles à partir d'une autre intention, la fusion de celles-ci oblige à construire une

nouvelle DSS (c). Les sections qui avaient comme cibles des intentions différentes ont maintenant les mêmes ce qui requière la définition d'une nouvelle DSS correspondant à cette situation.

Exemple

L'exemple de la Figure 108 montre la fusion des deux composants à travers la fusion des *intentions "Découvrir un but"*. Le composant obtenu contient la nouvelle intention et toutes les sections dont cette intention fait partie.

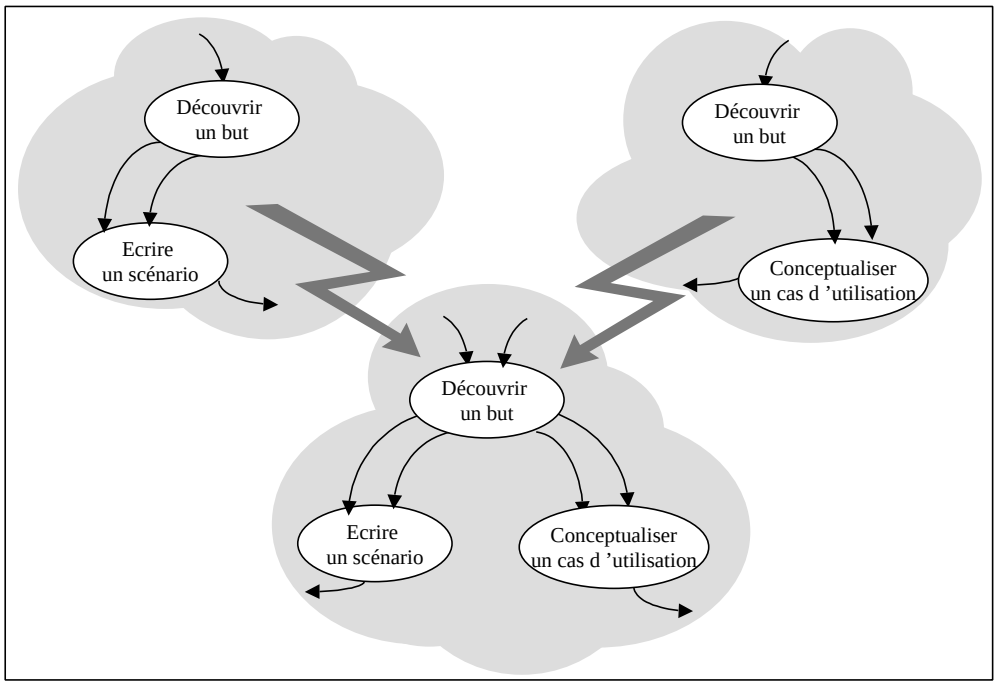


Figure 108 : Exemple d'application de l'opérateur `FUSIONNER_INTENTION`

1.2.2.4.2 FUSIONNER_SECTION

Motivation

Formalisme

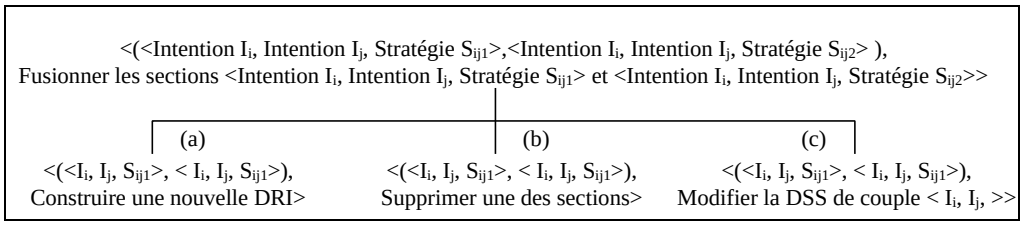
`FUSIONNER_SECTION C * I *I * S * S → C`

`FUSIONNER_SECTION(<`i_i`,`i_j`,`s_{ij1}`>,<`i_i`,`i_j`,`s_{ij2}`) = c ∪ <`i_i`,`i_j`,`s_{ij3}`> \ <`i_i`,`i_j`,`s_{ij1}`> ∪ <`i_i`,`i_j`,`s_{ij2}`>`

Démarche

Signature : <(<Intention I_i, Intention I_j, Stratégie S_{ij1}>,<Intention I_i, Intention I_j, Stratégie S_{ij2}>), Fusionner les sections <Intention I_i, Intention I_j, Stratégie S_{ij1}> et <Intention I_i, Intention I_j, Stratégie S_{ij2}>>

Corps :



Description

L'opérateur `FUSIONNER_SECTION` permet de fusionner deux sections de cartes différentes ayant la même intention source et la même intention cible. Les deux intentions correspondantes doivent être fusionnées au préalable. Les stratégies des deux sections doivent avoir une sémantique similaire.

La fusion des intentions peut amener à une situation où deux sections proposent la même démarche ou des démarches similaires. L'ingénieur de méthodes peut décider de supprimer une des sections si les démarches qu'elles proposent sont identiques ou de les fusionner si les démarches proposées sont similaires mais peuvent se compléter l'une l'autre.

La fusion de deux sections consiste à fusionner les directives DRI des sections initiales en construisant une nouvelle directive DRI (a). La nouvelle DRI est une directive choix qui propose comme alternatives les deux DRI initiales. La directive DSS doit aussi être modifiée en fonction du résultat obtenu (b). Le choix des stratégies précédentes est remplacé par celui de la nouvelle stratégie. Finalement, une des sections initiales doit être éliminée de la carte.

2. MESURES DE LA SIMILARITE

Des techniques de mesure de similarité ou de distance des schémas de conception ont été proposées dans la littérature. Ces mesures servent à évaluer la réutilisabilité des composants de schémas conceptuels [Castano 92], [Castano 93] et la sélection de ces composants [Jilani 97]. En général, ces approches estiment la proximité entre les entités des différents schémas conceptuels en termes des propriétés qu'elles ont en commun et des propriétés qui sont propres à chacune d'elles ainsi que les liens de dépendance avec d'autres entités [Castano 93]. La complexité globale des schémas est aussi prise en compte.

Bianco et al. proposent des métriques de similarité pour l'analyse des schémas des bases de données hétérogènes [Bianco 99]. Ces métriques sont basées sur la similarité sémantique et structurelle des éléments de schémas. Plusieurs autres approches de mesure de similarité ont été proposées pour aider à l'indexation [Diamantini 99], le regroupement et la sélection de données similaires [Papadopoulos 99]. Des métriques de similarité textuelle ont été appliquées à la recherche des documents textuels [Besançon 99]. Dans le domaine de la re-ingénierie des systèmes d'information, les mesures de distance sont utilisées pour évaluer les changements des modèles d'entreprise [Poels 2000].

Les mesures de similarité que nous proposons ici sont inspirées de celles de Castano [Castano 93] et Bianco [Bianco 99]. Contrairement à ces deux approches, la notre ne se restreint pas à la comparaison des schémas de données. Nous distinguons deux types de mesures, celles qui permettent de mesurer la

similarité des éléments des parties de produit des composants de méthodes et celles qui sont destinées à la comparaison de leurs directives, c'est-à-dire à leurs processus. Les deux sous-sections suivantes développent les mesures de similarité des éléments des parties de produit et des éléments de directives respectivement.

2.1 Mesures de similarité des éléments des modèles de produit

L'application des opérateurs nécessite souvent de mesurer d'abord les similarités des éléments de différents modèles de produit. Afin de pouvoir appliquer les opérateurs d'unification, de modification et de fusion (section 1.1.2) on est amené à comparer d'abord les concepts qui sont concernés par l'opérateur. Nous proposons de mesurer la similarité des concepts en prenant en compte deux paramètres : leur sémantique et leur structure [Castano 93], [Bianco 99].

Considération sémantique: L'affinité sémantique entre concepts est basée sur les objets du monde réel représentés par les concepts.

Considération structurelle: L'affinité structurelle prend en compte la structure des concepts et leurs liens avec d'autres concepts.

2.1.1 Similarité sémantique

Les parties de produit de différents composants de méthode ont souvent des concepts de mêmes noms mais représentant des objets différents ou, inversement, représentant les mêmes objets du monde réel et nommés différemment. Afin de pouvoir résoudre ces conflits de polysémie et de synonymie, nous introduisons dans notre approche une métrique permettant de mesurer la similarité sémantique des concepts.

La similarité sémantique entre deux concepts de modèles de produit est basée sur la similarité des noms de ces concepts. Nous appelons cette métrique *affinité de noms (AN)*.

Afin de pouvoir calculer le coefficient d'affinité des noms de concepts, nous proposons de définir d'abord tous les concepts dans un thésaurus. De plus, les relations suivantes doivent être définies entre les termes du thésaurus :

- SYN (Synonyme-de) définit la relation de synonymie entre deux termes t_i et t_j où $t_i \neq t_j$ qui sont considérés comme des synonymes. Par exemple, nous avons une relation de synonymie <*But SYN Objectif*>. SYN est une relation symétrique.

- HYPER (Hyperonyme-de) définit la relation de hyperonymie entre deux termes t_i et t_j où $t_i \neq t_j$ et le terme t_i a un sens plus général que le terme t_j. On dit que le terme t_i et l'hyperonyme du t_i. Par exemple, nous avons une relation d'hyperonymie <*Scénario HYPER Scénario d'exception*>. HYPER n'est pas une relation symétrique. La relation inverse à la relation HYPER est la relation d'hyponymie HYPO.

Le thésaurus des termes peut être vu comme un réseau où les termes sont des nœuds et les relations terminologiques sont des arcs entre les nœuds. Chaque relation terminologique R $\in$ {SYN , HYPER /HYPO } a un poids associé s_R. Par exemple, $s_{SYN} = 1$ et $s_{HYPER/HYPO} = 0.8$.

Finalement, le coefficient d'*affinité des noms* des éléments de modèles de produit est calculé de la manière suivante :

$$AN(n(e_{ij}), n(e_{kl})) = \begin{cases} 1 & \text{si} < n(e_{ij})\ SYN\ n(e_{kl}) > \\ s_{1R} * \ldots * s_{(m-1)R} & \text{si } n(e_{ij}) \rightarrow {}^{m}n(e_{kl}) \\ 0 & \text{sinon} \end{cases}$$

où $n(e_{ij}) \rightarrow {}^{m}n(e_{kl})$ indique que la longueur du chemin entre les termes e_{ij} et e_{kl} dans le thésaurus est m avec m ≥ 1, σ_{nR} indique le poids de la n-ième relation terminologique dans $n(e_{ij}) \rightarrow {}^{m}n(e_{kl})$.

Par exemple, le coefficient d'affinité des noms entre les concepts "But" et "Objectif" est égal à 1 (*AN(But, Objectif) = 1*) car ces deux termes sont définis comme des synonymes (<*But* SYN *Objectif*>).

Prenons comme exemple deux composants de méthodes, l'un issu de la méthode CREWS-*L'Ecritoire* et l'autre issu de la méthode OOSE. La Figure 109 représente les thésaurus de ces composants.

	Concept	**Propriétés structurelles**	**Liens avec les concepts**
CREWS-	Scénario	Label, Auteur	(composé de) Action, (association) Etat (initial), (association) Etat (final), (composant de) FB (généralisation de) Scénario normal, (généralisation de) Scénario exceptionnel
L'Ecritoire	Scénario exceptionnel		(spécialisation de) Scénario
	Scénario normal		(spécialisation de) Scénario
	But	Verbe, Cible, Source, Manière, Moyen, Destination, Bénéficier	(composant de) FB
	Action		(composant de) Scénario
	Agent	Nom_agent, Description	(association) Action
	FB		(composé de) Scénario (composé de) But

	Concept	**Propriétés structurelles**	**Liens**
OOSE	Scénario	Label, Nom, Auteur, Description	(composant de) Cas d'utilisation
	Scénario exceptionnel		(spécialisation de) Scénario
	Scénario normal		(spécialisation de) Scénario
	Cas d'utilisation	Nom, Description, Objectif	(association) Acteur,

			(composé de) Scénario
	Acteur	Nom_acteur, Description	(association) Cas d'utilisation
	....		

Figure 109 : Les thésaurus des méthodes OOSE et CREWS - *L'Ecritoire*

Supposons également, que les relations terminologiques entre les termes de ces thésaurus sont les suivantes : <*But* **SYN** *Objectif*>, <*Agent* **SYN** *Acteur*>, <*Scénario* **HYPER** *Scénario exceptionnel*>, <*Scénario normal* **HYPO** *Scénario* >, <*Cas d'utilisation* **HYPER** *Scénario*>, où $s_{SYN} = 1$ et $s_{HYPER/HYPO} = 0.8$.

Calculons maintenant le coefficient d'affinité de noms entre les concepts "But" et "Objectif", "Agent" et "Acteur", "Scénario" et "Scénario exceptionnel" et "Scénario exceptionnel" et "Scénario normal".

Puisque nous avons déjà défini que <*But* **SYN** *Objectif*>, alors *AN(But, Objectif) = 1*. De la même manière *AN(Agent, Acteur) = 1*.

En ce qui concerne les concepts de "Scénario" et "Scénario exceptionnel", le coefficient *AN(Scénario, Scénario exceptionnel) = 0.8* car <*Scénario* **HYPER** *Scénario exceptionnel*> *et* $s_{HYPER/HYPO} = 0.8$.

Finalement, le coefficient *AN(Scénario exceptionnel, Scénario normal) = 0.8*0.8 = 0.64* car <*Scénario* **HYPER** *Scénario exceptionnel*> et <*Scénario* **HYPER** *Scénario normal*>.

2.1.2 Similarité structurelle

Une fois les concepts comparés du point de vue sémantique, nous proposons de les comparer du point de vu de leurs structures. En effet, la similarité sémantique ne suffit pas à déterminer si deux concepts sont similaires car même si leur sémantique l'est, leur structure peut être différente selon le modèle auquel ils appartiennent.

La comparaison structurelle des concepts est basée sur le calcul des propriétés communes et des liens communs avec d'autres concepts. Nous mesurons la similarité structurelle à deux niveaux : celui des propriétés structurelles des concepts (ou des liens) et celui des concepts.

2.1.2.1 Affinité structurelle des propriétés (ASP)

Le coefficient *d'affinité structurelle des propriétés (ASP)* est calculé en deux étapes. On mesure dans un premier temps l'affinité des noms des propriétés et dans un deuxième temps la compatibilité des domaines de valeurs. Le coefficient d'affinité structurelle des propriétés de deux éléments des modèles de produit est calculé de la manière suivante :

$$ASP(p_{ij}, p_{kl}) = \begin{cases} \text{correspondance forte} & \text{si } AN(p_{ij}, p_{kl}) = 1 \text{ et } d(p_{ij}) = d(p_{kl}) \\ \text{correspondance faible} & \text{si } AN(p_{ij}, p_{kl}) \leq 1 \text{ et } d(p_{ij}) \text{ compatible avec } d(p_{kl}) \\ \text{pas de correspondance} & \text{si } AN(p_{ij}, p_{kl}) = 0 \text{ ou } d(p_{ij}) \neq d(p_{kl}) \end{cases}$$

Prenons l'exemple de la Figure 109 à nouveau. Soit le coefficient ASP entre les propriétés *"Nom_agent"* et *"Nom_acteur"* appartenant aux concepts *"Agent"* de CREWS-*L'Ecritoire* et *"Acteur"* de OOSE respectivement. Comme le montre la formule ci-dessus, afin de pouvoir mesurer ce coefficient il faut d'abord calculer le coefficient AN des deux propriétés puis comparer leurs domaines. Le coefficient *AN(Nom_agent, Nom_acteur) = 1* car les deux termes sont synonymes. En ce qui concerne les domaines de valeurs des deux propriétés, tous les deux sont de type "*chaîne de caractères"*, donc égaux. Par conséquent, suivant la formule de calcul ci-dessus :

ASP(Nom_agent, Nom_acteur) = correspondante forte.

2.1.2.2 Affinité structurelle des concepts (ASC)

Le coefficient *d'affinité structurelle de concepts (ASC)* permet de mesurer la similarité des concepts par rapport à leurs propriétés structurelles en calculant quel est le pourcentage de propriétés qui leur sont communes. La formule d'évaluation est la suivante :

$$\mathrm{ASC}(c_1, c_2) = \frac{2 * (\text{Nombre de propriétés communes de } c_1 \text{ et } c_2)}{\sum_{i=1}^{2} \text{Nombre de propriétés du } c_i}$$

On appelle *propriétés communes* les propriétés dont le coefficient ASP a comme valeur *"correspondance forte"* ou *"correspondance faible"*.

Prenons comme exemple les concepts *"Agent"*, du composant de CREWS-*L'Ecritoire*, et *"Acteur"*, du composant de OOSE, qui sont décrits à la Figure 109. Selon la formule proposée ci-dessus, il faut d'abord calculer le nombre de propriétés qui sont communes aux deux concepts. Comme le montre la Figure 109, les deux concepts ont chacun deux propriétés. Afin de pouvoir déterminer si ces propriétés sont communes, il faut calculer l'ASP pour chaque paire de propriétés.

Comme nous avons déjà vu à la section précédente, *ASP(Nom_agent, Nom_acteur) = correspondante forte*.

En ce qui concerne la deuxième paire des propriétés, nommées *"Description"*, des deux concepts, il est évident, que le coefficient ASP entre ces deux propriétés est aussi égal à *"correspondance forte"*.

Par conséquent, les deux concepts ont deux propriétés communes. Nous pouvons maintenant calculer leur coefficient ASC:

$$\mathrm{ASC}(\textit{Agent, Acteur}) = \frac{2 * (2 \text{ propriétés communes})}{2 \text{ propriétés } \textit{d'Agent} + 2 \text{ propriétés } \textit{d'Acteur}} = 1$$

Le résultat obtenu signifie que les deux concepts ont une sémantique (voir exemple de la section 2.1.1) et une structure similaire.

Prenons comme deuxième exemple le concept *"Scénario"* qui existe dans les deux composants présentés à la Figure 109. Le coefficient ASC pour ces deux concepts est le suivant :

$$\mathrm{ASC}(\mathit{Scénario}_{OOSE}, \mathit{Scénario}_{CREWS-L'Ecritoire}) =$$

$$\frac{2 * (2 \text{ propriétés communes})}{4 \text{ propriétés du } \mathit{Scénario}_{OOSE} + 2 \text{ propriétés du } \mathit{Scénario}_{CREWS-L'Ecritoire}} \approx 0.66$$

Les concepts ont deux propriétés communes : *"Label"* et *"Auteur"*, tandis que le *"Scénario"* de CREWS-*L'Ecritoire* en a quatre au total. Le coefficient ASC démontre que les deux concepts *"Scénario"* de CREWS-L'Ecritire et de OOSE ont une sémantique similaire mais que leur structure est différente.

2.1.2.3 Affinité d'adjacents des concepts (AAC)

Le coefficient *d'affinité d'adjacents de concepts (AAC)* permet de mesurer la similarité structurelle des concepts par rapport à leurs relations avec d'autres concepts du modèle de produit correspondant. Il calcule quel est le pourcentage de relations qui sont communes aux deux concepts de la manière suivante :

$$\mathrm{AAC}(c_1, c_2) = \frac{2 * (\text{Nombre de concepts adjacents communs aux } c_1 \text{ et } c_2)}{\sum_{i=1}^{2} \text{Nombre de concepts adjacents au } c_i}$$

Les *adjacents* d'un concept sont les concepts qui sont associés au concept initial par des liens d'association ou de composition.

Prenons le concept *"Scénario"* de la Figure 109 qui existe dans les deux composants de notre exemple. Pour mesurer le coefficient ASC entre le *"Scénario"* de CREWS-*L'Ecritoire* et le *"Scénario"* de OOSE, nous devons identifier les concepts qui sont adjacents à chacun de ceux-ci et compter ceux qui sont communs aux deux. Selon le thésaurus de la Figure 109, le *"Scénario"* de CREWS-*L'Ecritoire* a quatre concepts adjacents (*Action, Etat (initial), Etat(final), FB*) et le *"Scénario"* de OOSE en a un (*Cas d'utilisation*). Les deux concepts n'ont aucun adjacent commun. Par conséquent, leur coefficient ASC et le suivant :

$$\mathit{ASC}(\mathit{Scénario}_{OOSE}, \mathit{Scénario}_{CREWS-L'Ecritoire}) =$$

$$\frac{2 * (0 \text{ concepts adjacents communs})}{1 \text{ concept adjacent au } \mathit{Scénario}_{OOSE} + 4 \text{ concepts adjacents au } \mathit{Scénario}_{CREWS-L'Ecritoire}} = 0$$

Le résultat obtenu signifie que la similarité des deux concepts du point de vue de leurs relations avec d'autres concepts est strictement nulle. En effet, en ce qui concerne leurs relations avec les autres concepts, les deux notions de "Scénario" sont en effet différentes. Le *"Scénario"* de CREWS-*L'Ecritoire* a une structure plus complexe, il est composé d'un ensemble d'*"Actions"*, il est associé à un *"Etat initial"* et à un *"Etat final"*. Il fait partie d'un *"FB"* tandis que le *"Scénario"* de OOSE a une structure plus simple, réduite à une description textuelle non-structurée.

2.1.2.4 Affinité structurelle globale (ASG)

Le coefficient *d'affinité structurelle globale (ASG)* permet de mesurer la similarité structurelle des concepts de manière générale. Il est calculé par la formule suivante :

$$\boxed{\mathrm{ASG}(c_1, c_2) = \frac{\mathrm{ASC}(c_1, c_2) + \mathrm{AAC}(c_1, c_2)}{2}}$$

Comme le montre la formule ci-dessus, la mesure ASG est une superposition des deux mesures précédentes. En effet, la structure d'un concept dépend non seulement de ses propriétés structurelles mais aussi de ses relations avec d'autres concepts. L'affinité structurelle globale associe donc ces deux aspects dans un même coefficient qui mesure de manière générale la similarité structurelle des deux concepts. Il est basé sur les mesures ASC et AAC des concepts en question.

Nous avons déjà calculé dans les exemples précédents les coefficients ASC et AAC pour les concepts "Scénario" de CREWS-*L'Ecritoire* et de OOSE. En prenant en compte ces valeurs nous calculons le coefficient ASC de la manière suivante:

$$\mathrm{ASC}(Sc*_{OOSE}, Sc_{CE**}) = \frac{(ASC(Sc_{OOSE}, Sc_{CE}) + AAC(Sc_{OOSE}, Sc_{CE}))}{2} = \frac{(0.66 + 0)}{2} \approx 0.33$$

* *Sc = Scénario,*

***CE = CREWS − L'Ecritoire*

La valeur obtenue montre que les deux concepts *"Scénario"* sont différents du point de vue structurel.

2.2 Mesures de la similarité des éléments des cartes

Comme nous l'avons déjà dit précédemment, l'assemblage des composants consiste non seulement à assembler des parties de produit mais aussi des directives. Afin de pouvoir assembler deux directives nous devons évaluer leur similarité sémantique et leur similarité structurelle. La comparaison des modèles de processus est une nouveauté dans le domaine de l'ingénierie des processus.

Nous nous concentrons ici sur la comparaison des directives stratégiques exprimées sous forme de cartes. Puisque les deux éléments principaux d'une carte sont l'intention et la section (à laquelle on associe une directive de réalisation d'intention) nous utilisons deux types de mesures pour évaluer la similarité des intentions et des sections appartenant à deux cartes différentes. La première mesure la similarité sémantique des éléments des cartes et la deuxième évalue leur similarité structurelle.

2.2.1 Similarité sémantique

La mesure de la similarité sémantique permet de comparer les éléments de deux cartes du point de vue de leur sens car l'assemblage de cartes nécessite au préalable d'identifier les éléments similaires. Deux types d'éléments sont importants pour l'intégration des cartes : les intentions et les sections. Nous proposons en conséquence deux types de mesure pour évaluer leur similarité : l'affinité sémantique

des intentions et l'affinité sémantique des sections. Ces deux mesures sont détaillées dans la suite de cette section.

2.2.1.1 Affinité Sémantique des Intentions (ASI)

L'*Affinité Sémantique des Intentions* (ASI) permet de mesurer la proximité de sens de deux intentions. Puisqu'une intention est composée d'un verbe et d'une cible. Ces deux paramètres sont impliqués dans la mesure de la similarité sémantique des intentions. Si les verbes des deux intentions sont identiques, ou synonymes, et que les cibles des deux intentions sont identiques, ou synonymes, leur affinité sémantique est égale à 1, sinon elle est égale à 0. Nous réutilisons la relation de synonymie *SYN* entre les termes des composants définie à la section 2.1.1. L'affinité sémantique des intentions est définie de la manière suivante :

$$\mathrm{ASI}(i_i, i_j) = \begin{cases} 1 & \text{si } (i_i.\text{verbe SYN } i_j.\text{verbe}) \wedge (i_i.\text{cible SYN } i_j.\text{cible})) \\ 0 & \text{sinon} \end{cases}$$

Quand ASI(i_i, i_j) = 1, les deux intentions ont une sémantique similaire ce qui permet d'envisager leur fusion lors de l'intégration des cartes correspondantes. Dans le deuxième cas, les intentions ont une sémantique différente et ne doivent pas être fusionnées.

Supposons que nous ayons deux composants et que le premier ait une carte comprenant l'intention *"Découvrir un but"* et le deuxième une intention *"Découvrir un besoin"*. Pour évaluer le coefficient ASI entre ces deux intentions nous devons d'abord comparer leurs verbes puis leurs cibles. Puisque les deux intentions ont le même verbe *"Découvrir"*, il ne reste qu'à comparer leurs cibles qui sont *"But"* pour la première intention et *"Besoin"* pour la deuxième. Ces deux termes sont synonymes. Par conséquent, *ASI(i_1 : Découvrir un but, i_2 : Découvrir un besoin) = 1*.

Comme deuxième exemple, prenons les intentions *"Construire une classe"* et *"Construire un événement"*. Le coefficient ASI entre ces deux intentions est égal à zéro, *ASI(i_1 : Construire une classe, i_2 : Construire un événement) = 0*, car même si les verbes des deux intentions sont identiques leurs cibles sont différentes.

2.2.1.2 Affinité Sémantique des Sections (ASS)

L'*Affinité Sémantique des Sections* (ASS) permet de mesurer la proximité sémantique de deux sections. Rappelons qu'une section est définie par un triplet <intention source, intention cible, stratégie>. Les calculs du coefficient ASS sont basés sur l'évaluation de la mesure ASI entre les deux couples d'intentions : entre les deux intentions sources et les deux intentions cibles. Les stratégies sont comparées en appliquant la relation de synonymie. L'ASS est égale à 1 si l'ASI des deux intentions sources est égale à 1, l'ASI des deux intentions cibles est égale à 1 et si les expressions des deux stratégies sont identiques ou synonymes. Dans ce cas, on dit que les deux sections sont similaires sémantiquement ; sinon l'AAS est égale à 0 et on dit que les deux sections ont une sémantique différente. L'affinité sémantique des sections est définie de la manière suivante :

$$ASS(< i_i, i_j, s_{ij} >, < i_k, i_l, s_{kl} >) = \begin{cases} 1 \text{ si } ASI(i_i, i_k) = 1 \wedge ASI(i_j, i_l) = 1 \wedge s_{ij} \text{ SYN } s_{kl} \\ 0 \text{ sinon} \end{cases}$$

Supposons que nous ayons deux sections, *<i_i : Découvrir un but, i_j : Documenter un but, s_{ij} : Stratégie par scénarios>* et *<i_k : Découvrir un besoin, i_l : Documenter un besoin, s_{kl} : Stratégie linguistique>*, à comparer. Suivant la formule ci-dessus, nous devons mesurer d'abord l'ASI pour les deux couples d'intentions :

1. On a montré dans l'exemple de la section précédente que

   ASI(i_i : Découvrir un but, i_k : Découvrir un besoin) = 1.

2. *ASI(i_j : Documenter un but, i_l : Documenter un besoin) = 1* car les deux intentions ont le même verbe et leurs cibles sont synonymes (voir la section précédente).

Il nous reste à comparer les stratégies des deux sections. Nous pouvons dire que la *"stratégie linguistique"* est un synonyme de la *"stratégie par scénarios"*, car les deux stratégies proposent de documenter un but / besoin de manière textuelle. En conséquence de ces trois éléments, nous pouvons déduire que :

ASS(<i_i : Découvrir un but, i_j : Documenter un but, s_{ij} : Stratégie par scénarios>, <i_k : Découvrir un besoin, i_l : Documenter un besoin, s_{kl} : Stratégie linguistique>) = 1.

Prenons un autre exemple qui évalue la similarité des sections suivantes : *<i_i : Démarrer, i_j : Découvrir un but, s_{ij} : Stratégie par acteur>* et *<i_k : Démarrer, i_l : Découvrir un but, s_{kl} : Stratégie per objectif>*. Les intentions sources et les intentions cibles de ces sections sont identiques, leurs stratégies, par contre , sont différentes. Par conséquent,

ASS(<i_i : Démarrer, i_j : Découvrir un but, s_{ij} : Stratégie par acteur>, <i_k : Démarrer, i_l : Découvrir un but, s_{kl} : Stratégie per objectif>) = 0.

2.2.2 Similarité structurelle

Les mesures de similarité structurelle permettent de comparer la structure de deux cartes afin de pouvoir identifier les parties qui se recouvrent. Nous proposons deux types de mesures : la similarité structurelle des intentions et la similarité structurelle des sections.

2.2.2.1 Similarité Structurelle des Intentions (SSI)

La *Similarité Structurelle des Intentions* (SSI) permet de mesurer quel est le pourcentage d'intentions similaires dans deux cartes. Les calculs de la SSI sont basés sur la recherche des intentions similaires dans les deux cartes, c'est-à-dire sur les calculs d'ASI entre leurs intentions. Pour deux cartes c_1 et c_2, la SSI est calculée de la manière suivante :

$$SSI(c_1, c_2) = \frac{2 * \text{Le nombre des intentions similaires dans } c_1 \text{ et } c_2}{\sum_{i=1}^{2} \text{Le nombre des intentions dans la carte } c_i}$$

Si SSI est supérieure à 0, les cartes ont des intentions similaires qui peuvent être fusionnées lors de l'intégration des cartes. Si SSI est égale à 0, les cartes n'ont aucune intention en commun.

Prenons comme exemple deux composants de méthodes dont les cartes sont présentées à la Figure 110. Ces cartes ont deux intentions similaires. Tout d'abord l'intention *"Conceptualiser un scénario"* existe dans les deux cartes. Ensuite, la mesure ASI (section 2.2.1.1) montre que les intentions *"Découvrir un but"* de la première carte et l'intention *"Découvrir un besoin"* de la deuxième carte sont également similaires car leurs cibles *"But"* et *"Besoin"* représentent des produits synonymes. On ne prend pas en compte les intentions *"Démarrer"* et *"Arrêter"*.

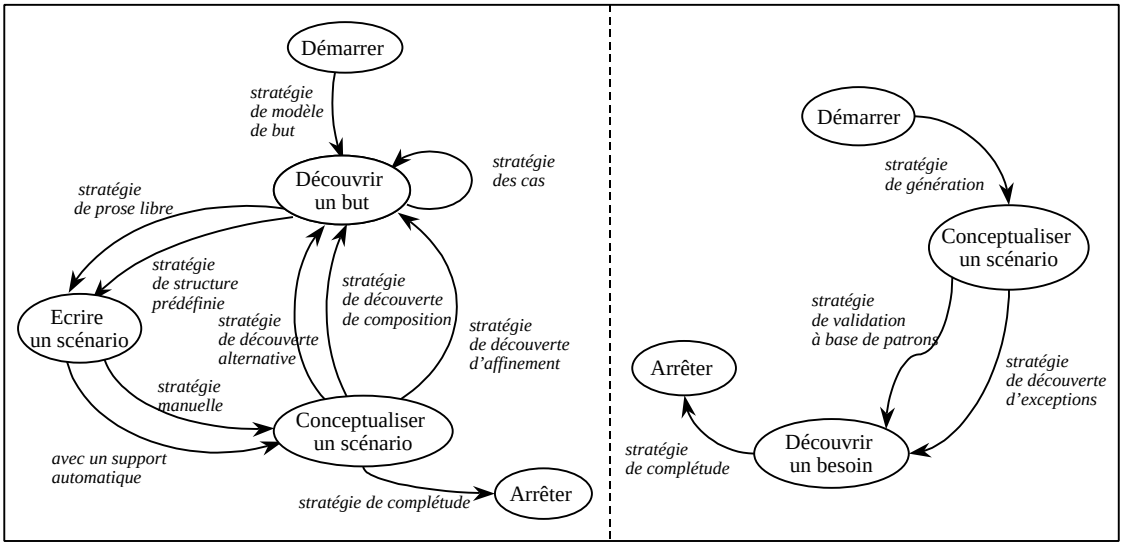


Figure 110 : Exemple de deux cartes

2.2.2.2 Similarité Structurelle des Sections (SSS)

La *Similarité Structurelle des Sections* (SSS) permet de mesurer le pourcentage de sections similaires dans les deux cartes. Soient c_1 et c_2 deux cartes de processus, la similarité structurelle des sections est définie de la manière suivante :

$$SSS(c_{1i}, c_2) = \frac{2 * \text{Le nombre des sections similaires dans } c_1 \text{ et } c_2}{\sum_{i=1}^{2} \text{Le nombre des sections dans la carte } c_i}$$

La *Similarité Structurelle Partielle des Sections* (SSPS) permet de mesurer le pourcentage des sections similaires par couple d'intentions. Soit c_1 et c_2 deux cartes de processus comportant les sections $<i_{1i},i_{1j}>$ et $<i_{2i},i_{2j}>$. La similarité structurelle partielle des sections est définie de la manière suivante :

$$SSPS(c_1 < i_{1i}, i_{1j} >, c_2 < i_{2k}, i_{2l} >) = \frac{2 * \text{Nb de sections similaires entre} < i_{1i}, i_{1j} > \text{et} < i_{2k}, i_{2l} >}{\text{Nb de sections entre} < i_{1i}, i_{1j} > + \text{Nb de sections entre} < i_{12k}, i_{2l} >}$$

Prenons encore une fois comme exemple les cartes présentées à la Figure 110 et mesurons la similarité structurelle partielle des sections entre les intentions *"Conceptualiser un scénario"* et *"Découvrir un but"* (en supposant que *"Découvrir un but"* soit identique à *"Découvrir un besoin"*). Il y a trois stratégies connectant ces intentions dans la première carte et deux dans la deuxième. Cependant, en comparant ces stratégies une a une, on s'aperçoit qu'elles sont toutes différentes. Par conséquent, la *SSPS (<Conceptualiser un scénario, Découvrir un but>, < Conceptualiser un scénario, Découvrir un besoin>) = 0.*

3. REGLES DE QUALITE

Dans cette section nous proposons un ensemble de règles qui permettent de valider la qualité du composant de méthode obtenu lors de l'assemblage des composants. Nous distinguons trois types de règles :

1. les règles de cohérence de l'assemblage des modèles de produit et des modèles de processus,
2. les règles de complétude du modèle de produit et du modèle de processus assemblé,
3. les règles de cohérence entre le modèle de produit et le modèle de processus obtenu.

Ces règles sont décrites dans les sous-sections suivantes.

3.1 Règles de cohérence

Les règles de cohérence servent pour garantir une application correcte des opérateurs d'assemblage. Puisque nous avons deux types d'opérateurs, les opérateurs d'assemblage des modèles de produit et les opérateurs d'assemblage des modèles de processus, les règles sont également divisées en deux sous-ensembles respectifs.

3.1.1 Règles de cohérence d'assemblage des modèles de produit

Les règles de cohérence d'assemblage des modèles de produit sont utilisées lors de l'assemblage des modèles de produit pour assurer l'application conforme des opérateurs d'assemblage. Certaines règles ont déjà été mentionnées dans la présentation des opérateurs.

Il est nécessaire de vérifier la terminologie des deux modèles de produit initiaux et de l'unifier, si nécessaire, avant leur assemblage. Ceci est pris en compte par les règles suivantes :

1. *Si deux concepts de modèles différents ont les mêmes noms mais des sémantiques et/ou des structures différentes, l'un de ces deux concepts doit être renommé avant l'assemblage des modèles de produit.*

2. *Si deux concepts des modèles de produits différents ont la même sémantique et une structure similaire mais des noms différents, l'un de ces deux concepts doit être renommé.*

L'assemblage des modèles de produit se fait soit par la fusion des éléments communs, soit par la connexion à travers des liens, soit par des concepts de connexion. Les règles suivantes aident à choisir et à appliquer les opérateurs nécessaires :

3. *Deux concepts de modèles différents ayant la même sémantique et une structure similaire peuvent être fusionnés dans le modèle intégré.*

4. *Deux concepts de modèles différents ayant la même sémantique mais des structures différentes ne peuvent pas être fusionnés dans le modèle intégré.*

5. *Un nouveau lien peut être ajouté dans le modèle intégré uniquement pour connecter les concepts d'origine des deux modèles de produit initiaux ou un concept créé au cours du processus d'assemblage avec le reste du modèle assemblé.*

6. *Un nouveau concept peut être ajouté dans le modèle intégré uniquement pour connecter les concepts d'origine des deux modèles de produit initiaux.*

3.1.2 Règles de cohérence d'assemblage des modèles de processus

Lors de l'assemblage des modèles de processus, il est nécessaire d'appliquer d'autres règles afin de pouvoir assurer l'application correcte des opérateurs d'assemblage correspondants. Certaines règles ont déjà été mentionnées dans la présentation des opérateurs.

Comme dans le cas des modèles de produit, il est nécessaire de valider la terminologie utilisée dans les cartes de processus des composants et d'apporter les modifications nécessaires afin de pouvoir les assembler correctement. Les règles suivantes doivent être appliquées :

1. *Deux intentions de cartes différentes ayant une sémantique similaire doivent avoir le même nom avant d'être intégrées.*

2. *Deux sections de cartes différentes ayant une sémantique similaire doivent avoir les mêmes expressions avant leur intégration.*

3. *S'il existe deux sections de cartes différentes, ayant la même intention source et la même intention cible, leurs stratégies doivent être nommées différemment si elles ont une sémantique différente.*

4. *S'il existe deux sections de cartes différentes ayant la même intention source et la même intention cible, leurs stratégies peuvent avoir le même nom si leurs sémantiques sont similaires.*

Le processus d'assemblage des modèles de processus est validé par les règles suivantes :

5. *Une nouvelle intention ne peut être ajoutée dans le modèle de processus assemblé que pour permettre la connexion de deux cartes n'ayant aucune intention commune.*

6. *Une nouvelle section ne peut être ajoutée dans le modèle de processus assemblé que pour permettre la connexion de deux cartes.*

3.2 Règles de complétude

Les règles de complétude permettent de vérifier si l'assemblage des modèles de produit et des modèles de processus est achevé. Dans cette section, nous proposons des règles qui doivent être vérifiées afin de terminer l'assemblage des composants.

3.2.1 Règles de complétude de modèle de produit final

Afin d'obtenir un modèle de produit complet on doit vérifier un certain nombre de règles sur le modèle de produit final.

On doit vérifier tout d'abord la terminologie du modèle de produit assemblé. Ceci est pris en compte par les règles suivantes :

1. *Chaque concept doit avoir un nom unique dans le modèle de produit final.*

2. *S'il existe deux liens entre deux concepts, ils doivent avoir des noms différents.*

3. *Les propriétés appartenant à des concepts différents peuvent avoir les mêmes noms.*

4. *Toutes les propriétés d'un concept doivent avoir des noms uniques.*

La structure du modèle de produit final doit satisfaire les règles décrites ci-dessous :

5. *Chaque concept dans le modèle de produit doit être connecté avec le reste du modèle par au moins un lien (association, composition ou spécialisation/généralisation).*

6. *Chaque lien d'un modèle de produit doit connecter au moins deux concepts.*

3.2.2 Règles de complétude de modèle de processus final

Dans cette section nous proposons des règles qui doivent être vérifiées pour assurer la complétude du modèle de processus final.

Un ensemble de règles permet de valider la terminologie du modèle de processus assemblé :

1. *Chaque intention doit avoir un nom unique dans la carte.*

2. *Si deux intentions sont connectées par plusieurs stratégies, ces stratégies doivent avoir des noms différents.*

3. *Il peut exister plusieurs stratégies ayant le même nom dans la carte.*

Il est également nécessaire de valider la structure de ce modèle :

4. *Chaque intention doit être connectée avec le reste du modèle de processus avec au moins une stratégie d'entrée et une stratégie de sortie.*

5. *Le modèle de processus intégré doit avoir une intention "Démarrer" et une intention "Arrêter".*

Le fait qu'une carte ait plusieurs différentes directives associées, l'association de ces directives à la carte du processus assemblé doit être vérifier par des règles de complétude correspondantes. Ces règles sont présentées ci-dessous :

6. *Chaque intention doit avoir une DSI associée dans le modèle de processus final.*

7. *Chaque ensemble de sections parallèles doit avoir une DSS associée dans le modèle de processus final.*

8. *Chaque section de la carte finale doit avoir une DRI associée.*

3.3 Règles de cohérence entre le modèle de produit et le modèle de processus assemblés

L'application des opérateurs d'assemblage sur les modèles de produit peut avoir un impact sur le modèle de processus assemblé et vice versa. Nous proposons des règles qui permettent de vérifier la cohérence entre le modèle de produit et le modèle de processus assemblés. Ces règles sont décrites ci-dessous.

Souvent, l'application des opérateurs d'assemblage pour assembler les modèles de produits implique des changements sur le modèle de processus assemblé. Par exemple, la suppression d'un concept, ou d'un lien, peut amener à une situation où la section ayant une directive utilisant (construisant ou modifiant) cet élément existe toujours dans le modèle de processus assemblé. Bien entendu, une telle section ne doit pas exister dans la carte finale. Ceci est vérifié grâce à la règle suivante :

1. *Chaque élément du modèle de produit doit être obtenu en exécutant le modèle de processus.*

Ceci est également vrai dans le cas inverse. Si, lors de l'assemblage des modèles de processus, une section est supprimée pour une raison quelconque et que l'élément de produit n'est plus utilisé par aucune autre section, il doit également être supprimé du modèle de produit final. La règle suivante prend en compte une telle situation :

2. *Chaque section de la carte finale doit manipuler au moins un élément du modèle de produit final.*

Plusieurs changements de noms peuvent être effectués lors de l'assemblage des modèles de produit ainsi que des ajouts de nouveaux concepts ou/et de nouveaux liens. Par conséquent, il est nécessaire de vérifier si toutes ces opérations ont été prises en compte dans le modèle de processus assemblé :

3. *Chaque partie de produit doit être nommé de la même manière dans le modèle de produit assemblé et dans les directives du modèle de processus qui le manipule.*

CHAPITRE 6

Modèle de processus d'assemblage de composants de méthodes

Ce chapitre est consacré à la présentation du modèle de processus d'assemblage que nous proposons dans cette thèse.

Nous nous intéressons dans ce travail aux différents processus d'assemblage de composants de méthode dans le but de construire une nouvelle méthode. Ces processus doivent guider l'ingénieur d'applications dans la construction d'une méthode situationnelle correspondant à un objectif identifié.

Pour représenter notre modèle de processus nous utilisons le méta-modèle de processus multi-démarche que nous avons présenté au Chapitre 3 en terme d'une directive stratégique. Ce méta-modèle utilise les concepts d'une carte de processus et de directives associées.

Dans la première section de ce chapitre nous proposons deux typologies d'assemblage sur lesquelles nous basons notre approche d'assemblage de composants de méthodes. La section 2 présente notre modèle de processus d'assemblage sous forme d'une directive stratégique, c'est-à-dire une carte avec des directives associées. Suivant le même raisonnement des processus modulaires que nous avons présenté dans la première partie de cette thèse, nous décomposons cette directive stratégique en composants que nous appelons des composants d'assemblage. Cette décomposition nous permet non seulement d'uniformiser la présentation de notre approche mais également de définir le contexte d'utilisation de chaque cas d'assemblage. Dans la section 3 nous décrivons en détails cinq principaux composants d'assemblage en proposant à chaque fois un exemple d'application. Finalement, dans la section 4 nous tirons les conclusions sur notre approche d'assemblage proposé dans ce chapitre.

1. TYPOLOGIE D'ASSEMBLAGE

Les tentatives de définition de processus d'assemblage actuelles se limitent à l'assemblage de composants de méthode qui concernent des activités complémentaires dans le processus de l'ingénierie d'un système d'information. Ce sont des activités qui se suivent les unes après les autres et qui, par conséquent, n'ont pas d'éléments communs [Brinkemper 98], [Punter 96], [Song 97]. Le processus d'assemblage se limite dans ce cas à l'identification des liens ou des concepts permettant de faire la connexion entre deux composants dans l'assemblage des parties de produit et de déterminer l'ordre d'exécution des processus correspondants dans l'assemblage des directives. Notre approche d'assemblage étend ce processus en prenant en compte l'assemblage de composants qui se recouvrent partiellement. Elle permet donc d'assembler des composants qui ont des concepts similaires dans leurs parties de produit et des intentions similaires dans leurs directives. Le processus reste applicable au cas d'assemblage des composants disjoints.

Dans notre approche nous distinguons deux modes d'assemblage :

1. assemblage *par association* et

2. assemblage *par intégration*.

Le premier mode concerne l'assemblage de composants qui n'ont aucun élément commun. En général dans ce cas, le produit résultat de l'application du premier composant est utilisé ensuite en tant que produit source dans le deuxième composant. Par exemple, le composant qui produit le modèle des cas d'utilisation et le composant qui produit le schéma objet du système n'ont aucun élément commun. En assemblant ces deux composants on obtient une méthode dans laquelle le deuxième composant utilise le résultat obtenu en appliquant le premier.

Le deuxième mode concerne l'assemblage de composants qui ont des éléments communs. L'assemblage de tels composants consiste à fusionner leurs éléments communs.

Une autre manière de classer les processus d'assemblage est basée sur l'objectif qu'ils permettent d'atteindre. L'ingénieur d'applications peut avoir des raisons différentes pour assembler des composants de méthodes. Nous en avons identifié trois:

1) définir une nouvelle méthode par assemblage de composants conformes aux exigences en cours,

2) compléter une méthode par une démarche alternative et

3) compléter une méthode par une fonctionnalité complémentaire.

Le premier objectif d'assemblage concerne la construction d'une nouvelle méthode situationnelle à partir d'un ensemble de composants. Le deuxième est d'enrichir le modèle de processus d'une méthode (ou d'un composant) par une nouvelle démarche capturée dans un autre composant. Les deux

composants sont complémentaires de point du vue du processus qu'ils proposent de mettre en œuvre. Finalement, le troisième objectif est d'enrichir une méthode (ou un composant) par une nouvelle fonctionnalité (un nouveau modèle) présentée sous forme d'un composant de méthode. Les deux méthodes dans ce cas sont complémentaires de point du vue du produit qu'elles construisent.

Les deux typologies sont orthogonales. La sélection des composants et leur assemblage dépendent de l'objectif à atteindre et des types de composants sélectionnés.

2. MODELE DE PROCESSUS D'ASSEMBLAGE DE COMPOSANTS DE METHODES

2.1 Présentation générale

En respectant les deux typologies présentées à la section précédente, nous proposons un modèle de processus d'assemblage de composant de méthodes présenté sous forme d'une carte et d'un ensemble de directives associées. Cette carte est décrite à la Figure 111.

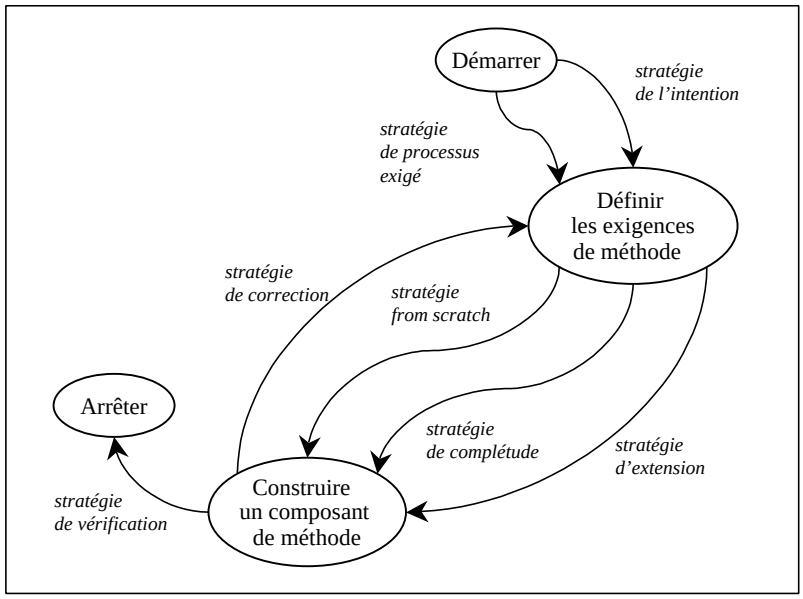


Figure 111 : Modèle de processus d'assemblage de composants de méthodes

Puisque toute méthode est également un composant de méthode (Chapitre 3), nous utilisons partout le terme de *composant de méthode* afin d'obtenir un modèle de processus universel, c'est-à-dire applicable sur tous les niveaux de granularité de composants (atomiques, agrégat et des méthodes toutes entières). Ainsi, la carte de processus d'assemblage (Figure 111) comporte deux intentions *Définir les exigences de composant de méthode* et *Construire un composant de méthode*.

Comme le montre la carte d'assemblage, tout processus d'assemblage commence par la définition des exigences sur le nouveau composant de méthode (nouvelle méthode). Pour satisfaire cette intention la

carte propose deux stratégies : *de processus exigé* et *de l'intention*. Ces deux stratégies sont basées sur les objectifs d'assemblage que nous avons définis dans la section précédente. La première stratégie est destinée à la définition des exigences pour une méthode qui n'existe pas et que l'ingénieur d'applications veut construire pour un projet particulier. Les exigences sont donc définies en fonction de type du projet et des préférences d'ingénieur d'applications. La deuxième stratégie concerne le cas où l'ingénieur d'applications voudrait appliquer une méthode particulière mais n'est pas complètement satisfait par cette méthode. Son objectif est soit de compléter la démarche de cette méthode si celle-ci n'est pas assez riche, soit d'étendre la méthode par une fonctionnalité complémentaire s'il en a besoin pour la conception du projet en cours et si la méthode choisie n'en possède pas. Dans les deux cas, l'ingénieur d'applications a une méthode (un composant) à la base de son processus d'assemblage.

La progression vers l'intention *Construire un composant de méthode* est possible par trois stratégies différentes : la *from scratch*, la *stratégie de complétude* et la *stratégie d'extension*. Les trois stratégies correspondent aux trois objectifs d'assemblage identifiés dans la section précédente de ce chapitre.

Le retour vers la définition des exigences est possible grâce à la stratégie de correction qui permet de modifier ou même redéfinir les exigences si nécessaire.

Finalement, la stratégie de vérification permet de terminer le processus d'assemblage en vérifiant si le composant obtenu répond aux exigences qui lui ont été imposées.

2.2 Présentation modulaire

Dans ce chapitre nous utilisons encore une fois la représentation modulaire des processus proposée dans la première partie de ce mémoire. Cette représentation nous permet de décrire chaque cas d'assemblage en terme d'un composant que nous appelons un *composant d'assemblage*. La décomposition du modèle de processus d'assemblage en composants est basée sur le processus de re-ingénierie de méthodes (Chapitre 4), plus exactement sur la partie de ce processus qui aide à décomposer une méthode modélisée par une carte en composants de méthode. Suivant ce processus toute DRI associée à une section de la carte peut être définie en terme d'un composant. Par conséquent, chaque section de notre carte d'assemblage (Figure 111) devient un composant d'assemblage. Les signatures de ces composants d'assemblage correspondent aux signatures des DRI associées aux sections de la carte et sont les suivantes :

CA1 : <(Description du projet), Définir les exigences de méthode avec la stratégie de processus exigé>

CA2 : <(Composant de méthode), Définir les exigences de méthode avec la stratégie de l'intention>

CA3 : <(Carte des exigences), Construire un composant de méthode avec la stratégie *from scratch*>

CA4 : <(Composant de méthode, Intention), Construire un composant de méthode avec la stratégie de complétude>

CA5 : <(Composant de méthode, Intention), Construire un composant de méthode avec la stratégie d'extension>

CA6 : <(Assemblage de composants de méthode), Définir les exigences de méthode avec la stratégie de correction>

CA7 : <(Assemblage de composants de méthode), Arrêter avec la stratégie de vérification>

Les deux premiers composants d'assemblage servent à définir les exigences sur le résultat souhaité.

Le premier composant (CA1) concerne le cas où l'ingénieur d'applications construit une nouvelle méthode pour un projet en cours. La construction d'une nouvelle méthode dans ce cas est dirigée par les exigences de l'ingénieur d'applications. Le composant aide à représenter l'ensemble des exigences par une démarche souhaitée exprimée sous forme d'une carte de processus que l'on appelle une *carte des exigences*. Ce composant devrait être directement suivi par le composant d'assemblage CA3.

Le deuxième composant (CA2) concerne le cas où l'ingénieur d'applications souhaite appliquer une méthode (un composant de méthode) particulière mais a besoin de la compléter par une démarche plus riche ou de l'étendre par une fonctionnalité complémentaire afin d'avoir toutes les fonctionnalités nécessaires à la conception du projet en cours. Dans les deux cas, le composant d'assemblage propose de définir ces exigences en terme d'une intention : identifier une intention dans la carte du composant initial dont la réalisation n'est pas assurée d'une manière satisfaisante et qui doit être complétée ou définir une intention à ajouter dans le modèle de processus du composant de méthode initial. Dans le premier cas, ce composant d'assemblage peut être suivi par le composant d'assemblage CA4, tandis que dans le deuxième cas il devrait être suivi par le composant d'assemblage CA5.

Les trois composant suivants (CA3-CA5) prennent en compte les trois objectifs d'assemblage définis dans la section précédente.

Le troisième composant d'assemblage (CA3) utilise l'ensemble des besoins (ou exigences) de l'ingénieur d'applications en termes de démarche. Nous faisons l'hypothèse que cette démarche est exprimée sous forme de carte. Les composants sont sélectionnés de telle manière qu'une fois assemblés ils doivent satisfaire l'ensemble des exigences, c'est-à-dire répondre à la carte. Ce cas peut utiliser les deux modes d'assemblage en fonction des composants sélectionnés. Le processus peut sélectionner des composants qui sont indépendants et des composants qui se recouvrent partiellement. Le résultat de l'assemblage est une nouvelle méthode construite à partir d'un ensemble de composants qui peut aussi être considérée comme un nouveau composant de méthode.

Le quatrième composant d'assemblage (CA4) propose d'enrichir la démarche d'un composant (ou d'une méthode) par une nouvelle démarche. Il peut être appliqué dans le cas où le modèle de processus du composant initial est trop pauvre ou inexistant. L'assemblage avec un autre composant permet d'enrichir le modèle de processus par une démarche définie dans un autre composant. Les composants à assembler doivent avoir des modèles de produit similaires. Le résultat de l'assemblage est le premier composant de méthode enrichi par la démarche du deuxième composant.

Le cinquième composant d'assemblage (CA5) permet d'enrichir un composant par une nouvelle fonctionnalité qui construit un nouveau produit ou modifie le produit du composant initial.

A tout moment de l'assemblage de composants de méthodes l'ingénieur d'applications peut décider de modifier ses exigences. Dans ce cas, il choisit le sixième composant d'assemblage (CA6) qui l'aide à redéfinir ses exigences.

Finalement, le dernier composant d'assemblage (CA7) permet d'arrêter le processus d'assemblage en vérifiant si le résultat obtenu répond complètement aux exigences initiales.

Les composants de CA3 à CA5 utilisent une des deux ou même les deux modes d'assemblage (par intégration et par association) suivant le type des composants à assembler. Dans les cartes de ces composants ces deux modes d'assemblage sont exprimés en termes des différentes stratégies d'assemblage. De plus, nous avons exprimé ces deux stratégies par des composants d'assemblage de niveau de granularité plus fin. Leurs signatures sont les suivantes :

CA3-1 : <(Composants Co1 et Co2), Assembler les composants Co1 et Co2 avec la stratégie d'intégration>

CA3-2 : <(Composants Co1 et Co2), Assembler les composants Co1 et Co2 avec la stratégie d'association>

Le composant CA3-1 suggère d'intégrer deux composants de méthode (ou deux méthodes) similaires de point de vue produit et/ou de point de vue processus. Les composants se recouvrent partiellement. Leur intégration permet d'obtenir un nouveau composant qui est plus riche par son produit et par son processus que les deux composants initiaux. Le mode *par intégration* est utilisé dans la directive de ce composant d'assemblage.

Le composant CA3-2 permet d'associer deux composants de méthode (ou deux méthodes) qui sont complémentaires du point de vue du produit qu'elles produisent. En général, ces composants correspondent à des vues différentes de la conception des systèmes. L'assemblage de tels composants permet d'obtenir une méthode complète qui traite tous les aspects de la conception d'un système. Le mode d'assemblage *par association* est utilisé par la directive de ce composant d'assemblage.

Dans la section suivante nous développons cinq composants d'assemblage les plus intéressants parmi ceux listés ci-dessus : les trois composants correspondants aux trois objectifs d'assemblage (CA3, CA4 et CA5) et les deux composants correspondants aux deux modes d'assemblage (CA3-1 et CA3-2).

3. COMPOSANTS D'ASSEMBLAGE

3.1 Assemblage dirigé par les exigences

3.1.1 Descripteur

Situation de réutilisation

Domaine d'application **:** Ingénierie de méthodes

Activité **:** Assemblage de composants de méthode

Intention de réutilisation

Construire une nouvelle méthode correspondant à la situation du projet

Objectif

L'objectif de ce type d'assemblage est de construire une nouvelle méthode répondant à la situation en cours à partir d'un ensemble de composants de méthodes. Aucune des méthodes existantes ne satisfait complètement les exigences de l'ingénieur d'applications, ou bien, les méthodes existantes doivent être adaptées à la situation du projet actuel. Notre proposition est de construire une nouvelle méthode répondant aux exigences à partir d'un ensemble de composants de différentes méthodes.

Type : Agrégat

3.1.2 Composant

Identifiant : CA3

Signature

Situation **:** Les exigences définies par un ingénieur d'applications sous forme d'une carte des exigences

Intention : Construire un composant de méthode avec la stratégie 'from scratch'

Directive

Type de directive : Stratégique

Représentation formelle : Le processus d'assemblage est représenté par la carte de la Figure 112. Les directives associées à la carte sont répertoriées dans le Tableau 2.

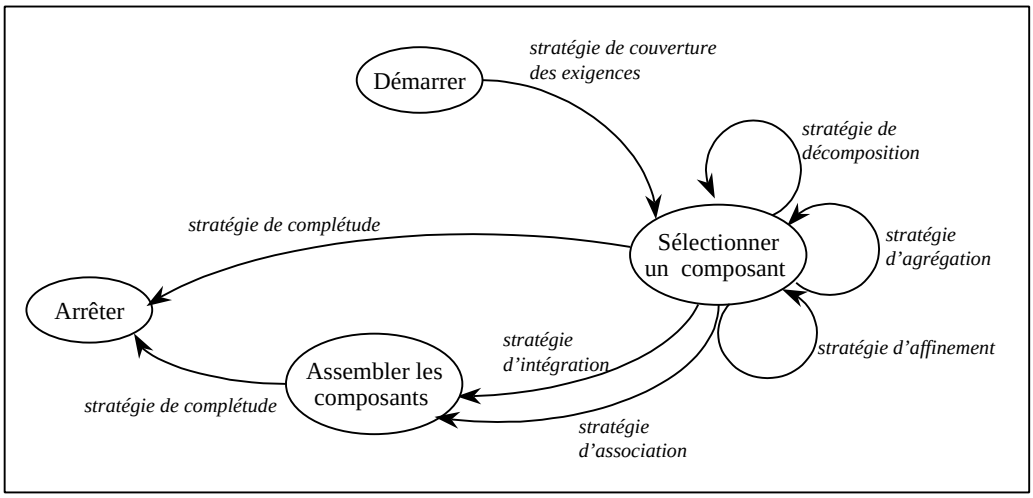
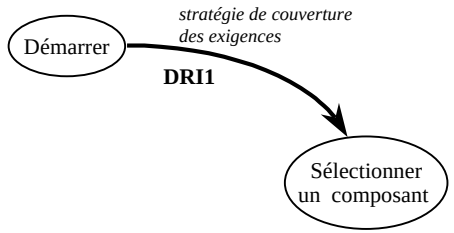
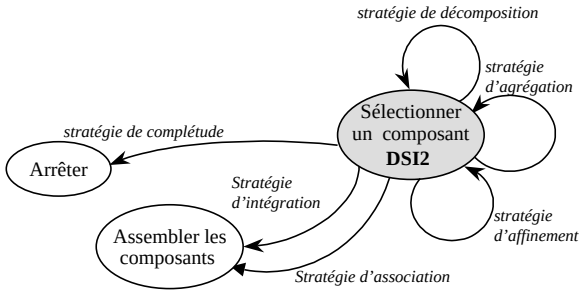
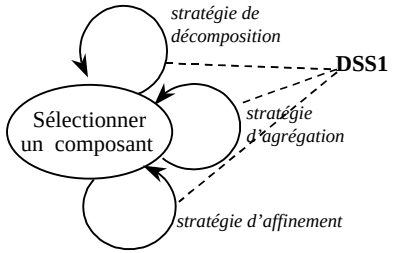
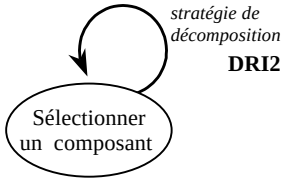
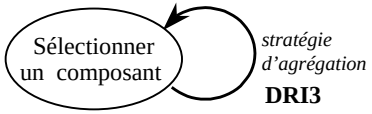
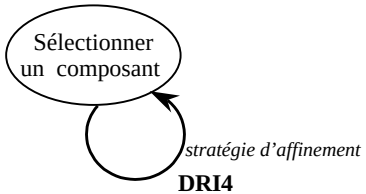


Figure 112 : Carte d'assemblage dirigé par les exigences

N°	Situation dans la carte de processus d'assemblage	Directive correspondant à la situation
1		**DRI1** : <(Carte des exigences), Sélectionner un composant avec la stratégie de couverture des exigences> (1) <(Carte des exigences), Sélectionner un composant Co> (2) <(Carte des exigences, Composant Co), Comparer le composant Co avec la carte des exigences> (3) <(Composant Co), Valider le composant Co> <(Composant Co), Sélectionner le composant Co> <(Composant C), Rejeter le composant Co> (1) Construire une requête de recherche dans la base de composants. (2) Mesurer la similarité structurelle des cartes avec les mesures de SSS et SSI (Chapitre 5). (3) Sélectionner le composant s'il couvre au moins une section de la carte des exigences, sinon le rejeter.
2		**DSI2** : <({ComposantCo}), Progresser de Sélectionner un composant> c1 <(Composant Co), Sélectionner DSS1: <(Composant Co), Progresser vers Sélectionner un composant>> c2 <(Composants Co1 et Co2), Sélectionner DSI2 :<(Composants Co1 et Co2), Progresser vers Assembler les composants Co1 et Co2>> c3 <(Composant Co), Sélectionner DRI7 : <(Composant Co) Arrêter avec la stratégie de complétude>> La directive propose trois choix pour progresser dans le processus d'assemblage : 1. Répéter l'intention "Sélectionner un composant" à l'aide de la DSS1 si c1. (situation 3) 2. Progresser vers l'intention "Assembler les composants" à l'aide de la DSS2 si c2. (situation 7) 3. Progresser vers l'intention "Arrêter" en exécutant la DRI7 si c3. (situation 10) Critères de choix : c1 = a1OU a2 ; c2 = a3 ; c3 = a4. Arguments : (a1) le composant est de type agrégat et sa directive est plus riche que ne le demande la carte des exigences ; (a2) le composant fait partie d'au moins un composant agrégat et il ne couvre pas toute la carte des exigences ;

<table>
<tr><td></td><td></td><td>(a3) au moins deux composants ont déjà été sélectionnés ;
(a4) la directive du composant est aussi riche que le processus exigé.</td></tr>
<tr><td>3</td><td> 
</td><td>DSS1 : <(Carte des exigences, Composant Co), Progresser vers Sélectionner un composant>
c1
c2
c3
<(Carte des exigences, Composant Co), Sélectionner DRI2 : <(Composant Co), Sélectionner un composant Co1 avec la stratégie de décomposition >>
<(Carte des exigences, Composant Co), Sélectionner DRI3 : <(Composant Co), Sélectionner un composant Co1 avec la stratégie d'agrégation >>
<(Carte des exigences, Composant Co), Sélectionner DRI4 :<(Composant Co), Sélectionner un composant Co1 avec la stratégie d'affinement >>
c1 : le composant est de type agrégat et sa directive est plus riche que ne le demande la carte des exigences ;
c2 : le composant fait partie d'au moins un composant agrégat et il ne couvre pas toute la carte des exigences ;
c3 : la directive du composant n'est pas assez riche.</td></tr>
<tr><td>4</td><td> 
</td><td>DRI2 : <(Carte des exigences, Composant Co), Sélectionner un composant Co1 avec la stratégie de décomposition>
(1) <(Composant Co), Identifier un sous-composant Co1>*
(2) <(Carte des exigences, Composant Co1), Comparer le composant Co1 avec la carte des exigences>
(3) <(Composant Co1), Valider le composant>
<(Composant Co1), Sélectionner le composant Co1>
<(Composant Co1), Rejeter le composant Co1>
(1) Sélectionner un sous-composant du composant Co (répéter pour tous les sous-composants du Co) ;
(2) Mesurer la similarité structurelle entre la directive du composant Co1 et la carte des exigences avec les mesures SSI et SSS (Chapitre 5).
(3) Sélectionner le composant s'il couvre au moins une section de la carte des exigences, sinon le rejeter.</td></tr>
</table>

5		**DRI3** : <(Carte des exigences, Composant Co), Sélectionner un composant Co1 avec la stratégie d'agrégation> (1) <(Composant Co), Identifier un super-composant Co1>* (2) <(Carte des exigences, Composant Co1), Comparer le composant Co1 avec la carte des exigences> (3) <(Composant Co1), Valider le composant> <(Composant Co1), Sélectionner le composant Co1> <(Composant Co1), Rejeter le composant Co1> (1) Sélectionner un super-composant Co1 du composant Co (le composant agrégat qui comporte le composant C. (répéter pour tous les super-composants de Co) ; (2) Mesurer la similarité structurelle entre la directive du composant Co1 et la carte des exigences avec les mesures SSI et SSS (Chapitre 5). (3) Sélectionner le composant s'il couvre au moins une section de la carte des exigences, sinon le rejeter.
6		**DRI4** : <(Carte des exigences, Composant Co), Sélectionner un composant Co1 avec la stratégie d'affinement> (1) <(Composant Co), Sélectionner un composant Co1 ayant la directive plus riche que Co>* (2) <(Carte des exigences, Composant Co1), Comparer le composant Co1 avec la carte des exigences> (3) <(Composant Co1), Valider le composant> <(Composant Co1), Sélectionner le composant Co1> <(Composant Co1), Rejeter le composant Co1> <(Composant Co), Identifier un composant Co1 ayant la même intention que Co>* <(Composants Co et Co1), Comparer les parties de produit de Co et de Co1>* <(Composants Co et Co1), Comparer les directives de Co et de Co1>* <(Composants Co1), Valider le composant Co1> (1) Sélectionner un Co1 ayant la même intention de signature que celle de Co mais avec une manière différente. Mesurer la similarité des parties de produits de Co et Co1 avec la mesure ASG. Mesurer la similarité des directives avec SSI et SSS. Si les parties des produit de Co et de Co1 sont similaires et la directive de Co1 est plus riche que celle de Co, sélectionner le Co1, sinon le rejeter. (2) Mesurer la similarité structurelle entre la directive du composant Co1 et la carte des exigences avec les mesures SSI et SSS (Chapitre 5). (3) Sélectionner le composant s'il couvre au moins une section de la carte des exigences, sinon le rejeter

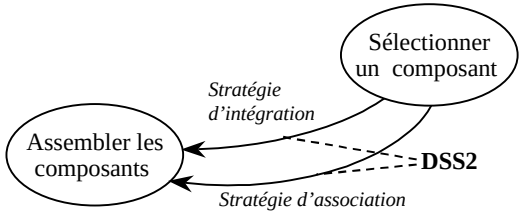
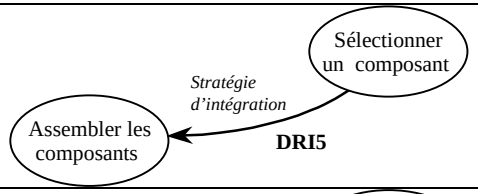
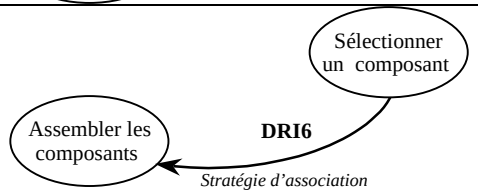
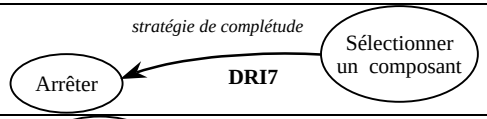
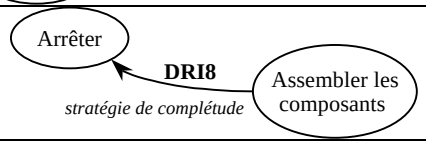
7		**DSS2** : <(Composant Co1 et Co2), Progresser ver Assembler les composants > c1 : <(Composant Co1 et Co2), Sélectionner DRI5 : <(Composant Co1 et Co2), Assembler les composants Co1 et Co2 avec la stratégie d'intégration>> c2 : <(Composant Co1 et Co2), Sélectionner DRI6 : <(Composant Co1 et Co2), Assembler les composants Co1 et Co2 avec la tratégie d'association>> Critères de choix : c1 : les composants Co1 et Co2 se couvrent partiellement c2 : les composants Co1 et Co2 n'ont aucun élément commun.
8		**DRI5** : <(Composants Co1 et Co2), Assembler les composants Co1 et Co2 avec la stratégie d'intégration> Cette directive est un composant d'assemblage CA3-1 que nous présentons à la section 3.4.
9		**DRI6** : <(Composants Co1 et Co2), Assembler les composants Co1 et Co2 avec la stratégie d'association> Cette directive est un composant d'assemblage CA3-2 que nous présentons à la section 3.5.
10		**DRI7** : <(Composant Co), Arrêter avec la stratégie de complétude> Arrêter le processus d'assemblage si la directive du composant couvre toute la carte des exigences.
11		**DRI8** : <(Composant assemblé), Arrêter avec la stratégie de complétude> Arrêter le processus d'assemblage si la directive du composant assemblé couvre toute la carte des exigences.

Tableau 2 : Signatures des directives associées à la carte de processus d'assemblage conforme aux exigences

Description

La construction d'une nouvelle méthode dans ce cas est dirigée par les exigences de l'ingénieur d'applications. L'ensemble des exigences correspondant à la démarche souhaitée est représenté par une carte de processus que l'on appelle une *carte des exigences*. Le composant d'assemblage aide l'utilisateur à sélectionner dans la base de composants de méthode un ou plusieurs composants correspondants à une ou plusieurs sections de la carte des exigences. Si plusieurs composants ont été sélectionnés, la carte d'assemblage aide à les assembler en appliquant les opérateurs d'intégration définis au Chapitre 5. La méthode obtenue est une nouvelle méthode plus adaptée à la situation en cours. De plus, si la méthode fait ses preuves lors de son application, elle peut être définie comme un nouveau composant de méthode de type agrégat et, par la suite, réutilisée en tant que telle ou en agrégation avec d'autres composants. En fonction du type de composants sélectionnés, les deux modes d'assemblage (intégration et association) peuvent être combinés pour construire une nouvelle méthode.

Le processus de définition d'une méthode par assemblage des composants selon les exigences que nous proposons dans ce composant d'assemblage est représenté par une directive stratégique sous forme d'une carte. Cette carte est illustrée à la Figure 112. A chaque situation de la carte on associe une directive appropriée (DRI, DSI ou DSS) afin de guider l'ingénieur d'applications dans la progression dans la carte et dans la réalisation des intentions. Ces directives sont répertoriées dans le Tableau 2. Nous développons maintenant chaque situation de manière informelle.

Situation 1 : Pour démarrer le processus d'assemblage à partir de la carte des exigences, l'ingénieur d'applications n'a qu'une seule possibilité de progresser vers l'intention *"Sélectionner un composant"* avec la *"stratégie de couverture des exigences"* en exécutant la directive associée (DRI1). Les composants de méthodes sont sélectionnés et comparés à la carte des exigences avec une technique de mesure de similarité des cartes. On cherche dans la base de méthodes les composants qui couvrent la plus grande partie de la carte des exigences. Un composant peut couvrir toute cette carte ou seulement une de ses sections. La technique de mesure de similarité des différents éléments des cartes permet de vérifier si le composant sélectionné dans la base de composants de méthode correspond aux besoins définis par la carte des exigences. Les stratégies de *décomposition, d'agrégation* et *d'affinement* aident à sélectionner dans la base le composant le mieux adapté à la situation courante (Situation 3).

Situation 2 : Apres avoir sélectionné un ou plusieurs composants, l'ingénieur d'applications est guidé par la DSI2 pour progresser dans la construction de nouveau composant de méthode en lui proposant trois possibilités :

1. rester sur la même intention *"Sélectionner un composant"* et affiner la sélection d'un composant en décomposant le composant sélectionné préalablement (s'il est de type agrégat) ou en cherchant des agrégats dont celui-ci est une partie ou même en redéfinissant la requête de la recherche avec des nouveaux paramètres,
2. assembler les composants sélectionnés en progressant vers l'intention *"Assembler les composants"*,

3. vérifier si le composant sélectionné ne couvre pas déjà toute la carte des exigences en progressant vers l'intention *"Arrêter"*.

La directive DSI1 associée à l'intention "*Sélectionner un composant*" aide l'ingénieur d'applications à faire son choix en lui proposant des arguments pour chacune des possibilités offertes.

Situation 3 : Admettons que l'ingénieur d'applications décide d'affiner la sélection d'un composant en restant sur l'intention "*Sélectionner un composant*". Ce choix est guidé par trois stratégies. La directive de sélection de stratégies DSI1 aide l'ingénieur d'applications à faire son choix parmi l'ensemble de stratégies proposées.

La *stratégie de décomposition* peut être sélectionnée si le composant de la situation est de type agrégat et si la carte de processus recouvre la carte des exigences mais contient des sections qui ne sont pas demandées dans cette carte. Cette alternative sélectionne la DRI2 qui permet de décomposer un composant en sous-composants. Elle est détaillée à la situation 4 du Tableau 2.

La *stratégie d'agrégation* s'applique lorsque la carte du composant sélectionné correspond exactement à une partie de la carte des exigences. La DRI3 sélectionnée suggère de vérifier si le composant n'appartient pas à un agrégat qui couvrirait une partie plus importante de la carte des exigences. Elle est détaillée à la situation 5 du Tableau 2.

La *stratégie d'affinement* propose de chercher d'autres composants ayant un modèle de produit similaire mais proposant un autre processus de développement et de les intégrer pour obtenir un processus encore plus riche que les deux processus initiaux. Cette alternative est réalisée par la DRI4 qui est détaillée à la situation 6 du Tableau 2.

Situation 4 : La DRI2 associée à la section <*Sélectionner un composant, Sélectionner un composant, Stratégie de décomposition*> peut être appliquée dans la situation où le composant sélectionné au préalable est un composant agrégat. La directive propose de vérifier si un de ses sous-composants ne correspond pas aux exigences de la méthode en cours de construction. Si c'est le cas, le sous-composant est sélectionné et le composant agrégat est éliminé, sinon, l'ingénieur d'applications peut décider d'éliminer le composant agrégat et d'en chercher un autre.

Situation 5 : La DRI3 associée à la section <*Sélectionner un composant, Sélectionner un composant, Stratégie d'agrégation*> peut être appliquée lorsque le composant sélectionné au préalable fait partie d'un ou plusieurs composants de type agrégat. L'ingénieur d'applications peut sélectionner un par un ces composants agrégat et vérifier si l'un d'eux ne recouvre pas une partie plus importante de la carte des exigences que le composant initial.

Situation 6 : Si le processus du composant sélectionné au préalable n'est pas assez riche, la DRI4 associée à la section <*Sélectionner un composant, Sélectionner un composant, Stratégie d'affinement*> propose de sélectionner un autre composant équivalent du point de vue du produit mais différent du point de vue du processus en redéfinissant les paramètres de la recherche dans la base de composants.

Situation 7 : Après avoir sélectionné au moins deux composants, l'ingénieur d'applications peut décider de les assembler (progresser vers l'intention *"Assembler les composants"*). La carte d'assemblage (Figure 112) propose deux stratégies pour assembler les composants : la *stratégie d'intégration* et la *stratégie d'association*. Le choix de la stratégie est guidé par la DSI2 qui propose de progresser avec la première stratégie si les composants se recouvrent partiellement et de progresser avec la deuxième s'ils n'ont aucun élément commun. La première alternative de la DSI2 sélectionne la DRI5 qui est détaillée à la situation 8 et la deuxième alternative sélectionne la DRI6 qui est présentée à la situation 9 du Tableau 2.

Situation 8 : Si les deux composants se recouvrent partiellement, la DRI5 associée à la section *<Sélectionner un composant, Assembler les composants, Stratégie d'intégration>* de la carte d'assemblage suggère à l'ingénieur d'applications d'appliquer le composant d'assemblage CA3-1 : *<(Composants Co1 et Co2), Assembler les composants Co1 et Co2 avec la stratégie d'intégration>*. Ce composant est développé à la section 3.4 de ce chapitre.

Situation 9 : Si les deux composants sélectionnés n'ont aucun élément commun, l'ingénieur d'applications est invité par la DRI6 à appliquer le composant d'assemblage CA3-2 : *<(Composants Co1 et Co2), Assembler les composants Co1 et Co2 avec la stratégie d'association>*. Ce composant est présenté à la section 3.5 de ce chapitre.

Situation 10 : Si le composant sélectionné en situation 2 est assez important et couvre pratiquement toute la carte des exigences, l'ingénieur d'applications peut décider d'arrêter le processus d'assemblage en déclarant que le composant sélectionné correspond à la méthode demandée.

Situation 11 : Après avoir assemblé au moins deux composants, l'ingénieur d'applications peut vérifier si le composant de méthode en construction ne couvre pas déjà la carte des exigences. Si c'est le cas, il décide d'arrêter le processus d'assemblage sinon il doit compléter en sélectionnant d'autres composants dans la base jusqu'à ce qu'il obtienne une méthode correspondant aux exigences, c'est-à-dire, dont la carte de processus couvre la carte des exigences.

3.1.3 Exemple de la construction d'une nouvelle méthode par assemblage de composants

Prenons comme exemple le cas de la construction d'une méthode permettant de découvrir les besoins fonctionnels du système sous forme d'une hiérarchie de buts, de les conceptualiser par des moyens linguistiques (écriture et analyse des différents types de scénarios), de valider les besoins d'une manière animée ou/et par des scénarios de validation et finalement, de les documenter.

Supposons que l'on ait appliqué le composant d'assemblage CA1 et que l'on ait défini la carte des exigences pour la nouvelle méthode décrite à la Figure 113.

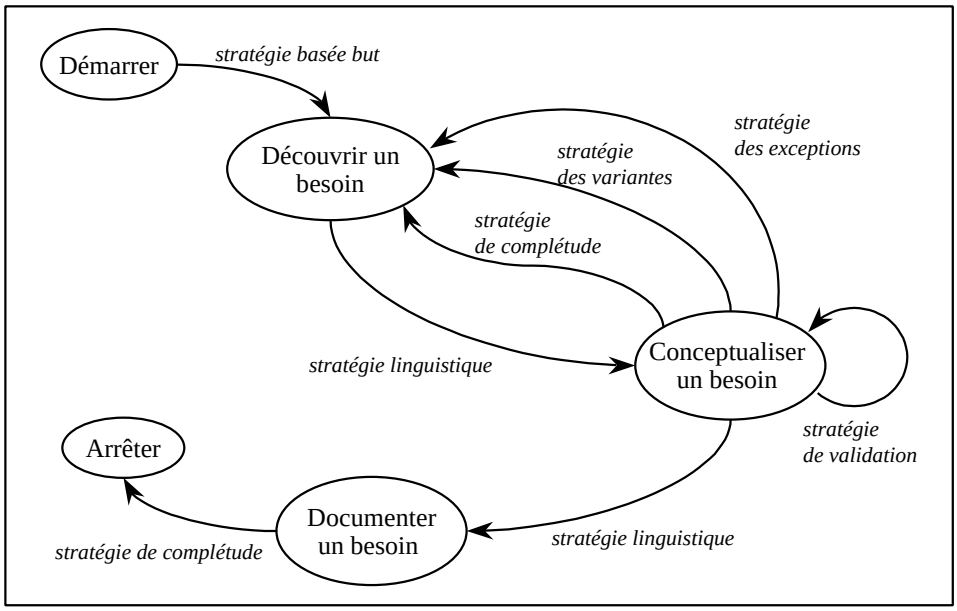


Figure 113 : Exemple d'une carte des exigences

La première étape consiste à sélectionner dans la base de composants de méthodes un ou plusieurs composants dont la carte de processus pourrait remplacer une partie de la carte des exigences ou même la carte des exigences tout entière. La requête de la sélection de composants est basée sur les valeurs des signatures et des descripteurs des composants. Par exemple, on sélectionne les composants dont les descripteurs contiennent les valeurs suivantes : le domaine d'application = *Systèmes d'information*, l'activité de conception = *l'ingénierie des besoins*, l'intention = *découvrir les besoins fonctionnels du système* et la situation de composant comporte la *description du problème*. Supposons que la liste des signatures des composants correspondants à la requête est la suivante :

1. <(Description du problème), Découvrir des couples But/Scénario suivant l'approche CREWS-*L'Ecritoire*>
2. <(Description du problème), Découvrir un but avec la stratégie de modèle de but>
3. <(Description du problème>, Construire un modèle des cas d'utilisation avec la stratégie de OOSE>
4. <(Description du problème>, Découvrir un cas d'utilisation avec la stratégie par acteur>

En analysant les composants trouvés on s'aperçoit que le composant N°1 est un composant agrégat et que le composant de N°2 fait partie du cet agrégat. Par conséquent, on choisit l'agrégat comme étant le plus complet et on le compare à la carte des exigences. La Figure 114 illustre comment la carte de ce composant (partie grise de la figure) répond à la partie de la carte des exigences concernant la découverte des besoins sous forme des buts et la conceptualisation des besoins sous forme des scénarios. On applique la mesure de similarité SSI *(Chapitre 5)* sur les deux cartes. Le coefficient obtenu *SSI(Carte des exigences, Carte* CREWS-*L'Ecritoire) = 2* 2 intentions communes / 6 intentions aux total = 2/3* montre que la plus grande partie de la carte des exigences est couverte par la carte du composant sélectionné. Pour préciser la similarité des cartes on vérifie si les sections comportant des intentions similaires proposent également des stratégies similaires. On applique la mesure SSPS sur des couples d'intentions qui existent dans les deux cartes *(Chapitre 5)*. Par exemple, *SSPS (Carte des exigences <Conceptualiser un besoin, Découvrir un besoin>, Carte de* CREWS-*L'Ecritoire <Conceptualiser un scénario, Découvrir un but>) = 2*2 sections communes / 6 sections*

au total) = 2/3 ce qui veut dire que la carte du composant sélectionné couvre bien une partie des sections de la carte des exigences.

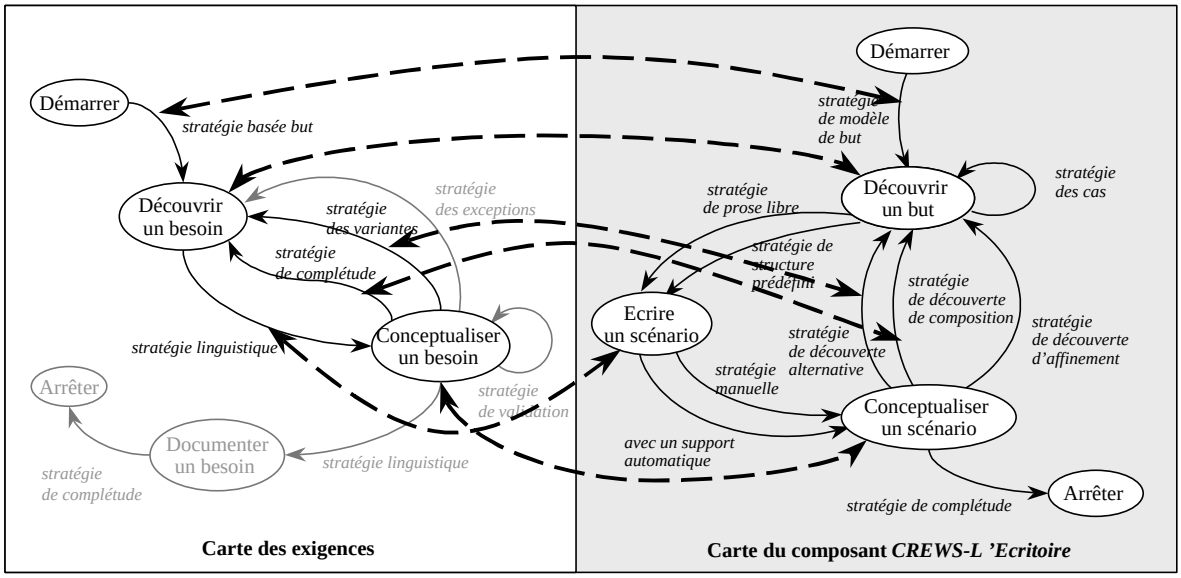


Figure 114 : Similarités entre la cartes des exigences et la carte du composant CREWS-*L'Ecritoire*

Le composant N°3 concerne également la conceptualisation des besoins sous forme des cas d'utilisation composés de scénarios, mais la démarche capturée dans ce composant est moins riche que celle du composant N°1. On ne s'intéresse qu'à la directive concernant la découverte des cas d'utilisation par les biais des acteurs. En décomposant le composant agrégat N°3 on choisit un de ses sous-composant qui a été déjà sélectionné en N°4.

Les deux composants ont le même objectif : *découvrir des besoins fonctionnels du système*. Par conséquent, pour les assembler on doit utiliser l'assemblage de type *par intégration*. Dans la carte d'assemblage de la Figure 112 on progresse avec la stratégie *"par intégration"* afin d'assembler les deux composants. Le résultat de cet assemblage est présenté à la Figure 115. La démarche concernant cet assemblage est contenue dans le composant d'assemblage *<(Composants Co1 et Co2), Assembler les composants Co1 et Co2 avec la stratégie d'intégration>* présenté à la section 3.4. L'assemblage de ces deux composants fait partie de l'exemple utilisé à cette section.

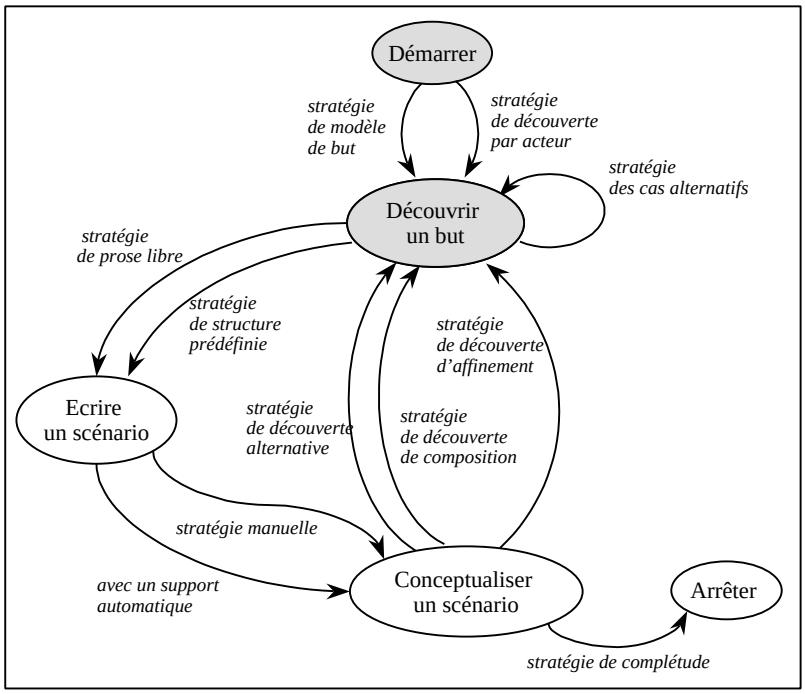


Figure 115 : Résultat d'assemblage de composants

Puisque l'assemblage obtenu ne couvre qu'une partie de la carte des exigences on décide de recommencer la sélection des composants dans la base de méthodes en cherchant à compléter le processus de découverte des buts et de leur conceptualisation. On construit une nouvelle requête avec les paramètres suivants : le domaine d'application = *Systèmes d'information*, l'activité de conception = *l'ingénierie des besoins*, l'intention de descripteur = *découvrir les besoins fonctionnels du système*, la situation du composant = *un but* ou *un besoin*, l'intention de composant = *découvrir un but* ou *découvrir un besoin* ou *conceptualiser un scénario*. Supposons que le résultat de la requête soit le suivant :

5. <(But, Scénario), Découvrir un but affiné avec la stratégie de découverte d'affinement>
6. <(But, Scénario), Découvrir un but complémentaire avec la stratégie de découverte de composition>
7. <(But, Scénario), Découvrir un but alternatif avec la stratégie de découverte alternative>
8. <(But), Découvrir un but avec la stratégie des cas alternatifs>
9. <(Besoin), Découvrir un besoin exceptionnel par l'analyse des erreurs humaines>

Les composants N°5 à 8 sont des sous-composants du N°1, ils font déjà partie de notre assemblage. Le composant N°9 est intéressant par son processus de découverte des exceptions causées par les erreurs humaines ce qui n'est pas couvert par le composant N°1. Ce composant aide à générer des scénarios pour un besoin donné à partir desquels il permet de découvrir des exceptions causés par des erreurs humaines ou de valider le besoin en appliquant des patrons de validation [Sutcliffe 98a], [Sutcliffe 98b], [Sutcliffe 99], [Maiden 98a], [Maiden 98b], [Maiden 98c]. Sa carte est représentée à la Figure 116.

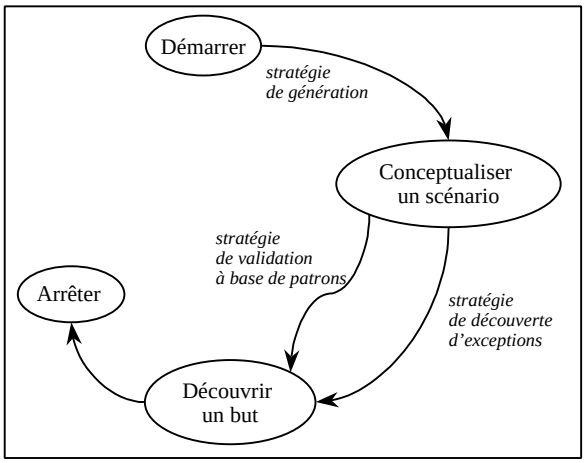


Figure 116 : Carte du composant N°9

On sélectionne le composant N°9 et on l'assemble avec le résultat obtenu précédemment. Puisque ce composant dessert le même objectif, c'est-à-dire la découverte des besoins, sa carte de processus contient les mêmes intentions, la stratégie à choisir pour progresser vers l'intention *Assembler les composants* dans la carte de l'assemblage (Figure 112) est toujours la *stratégie d'intégration* qui fait appel au composant d'assemblage de la section 3.4. Le résultat obtenu est illustré à la Figure 117.

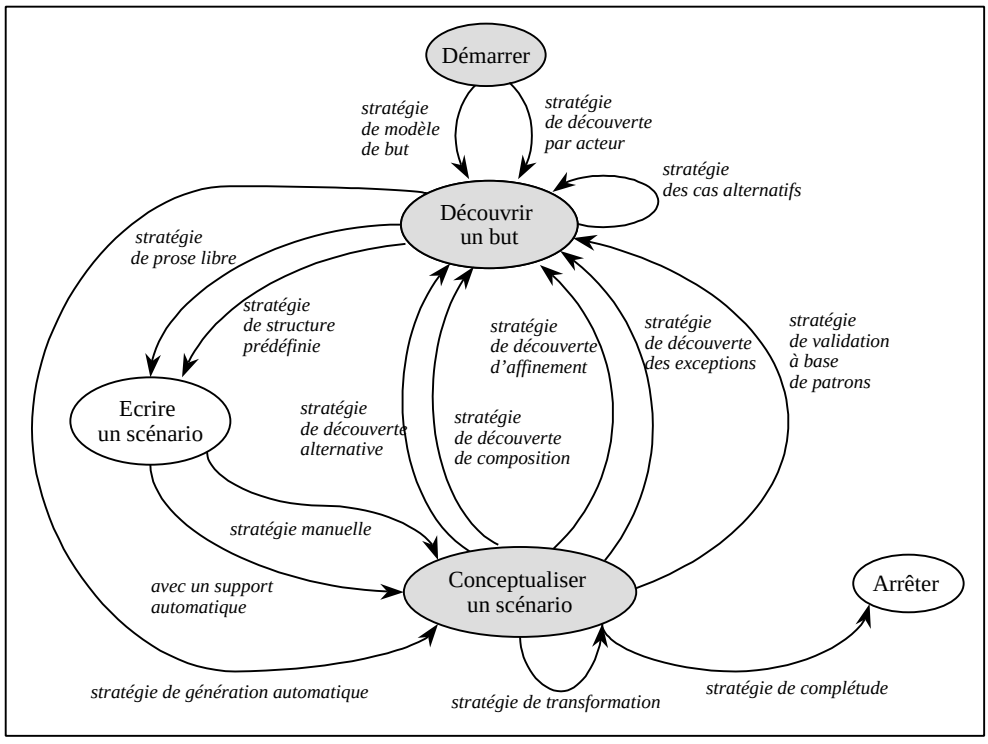


Figure 117 : Résultat d'assemblage des composants N° 1, 3 et 9

Après avoir assemblé les trois composants on fait appel à la directive de progression DSS1 (Tableau 2) qui suggère de progresser vers la sélection d'autres composants de méthodes pour couvrir le reste de la carte des exigences. On construit alors une nouvelle requête qui cherche dans la base des composants dont le domaine d'application est *Système d'information* et l'activité de conception est *l'ingénierie des*

besoins et dont l'intention de descripteur est *valider les besoins*. La liste des signatures des composants sélectionnés est la suivante:

10. <(Scénario, Besoin), Valider un besoin avec la stratégie d'animation>
11. <(Scénario), Découvrir un besoin avec la stratégie de validation à base de patrons>

Le composant N°11 est un sous composant du N°9, il est déjà inclus dans notre assemblage, tandis que le composant N°10 propose une autre manière de validation des besoins découverts préalablement, celle d'animation des scénarios. Ce composant transforme les scénarios obtenus préalablement en des spécifications *Albert* qui sont animées ensuite en utilisant un outil animateur et validées par les futurs utilisateurs du système [Heymans 98], [Dubois 98]. La carte de processus de ce composant est décrite à la Figure 141 pour illustrer le cas d'assemblage par association.

L'objectif de ce composant n'est pas le même que celui des composants sélectionnés précédemment, il est plutôt complémentaire. Le composant utilise des scénarios en entrée et les anime afin de valider les besoins qu'ils décrivent. Par conséquent, l'assemblage avec les composants sélectionnés précédemment n'est pas de même type. Dans ce cas on choisit la *stratégie d'association* pour progresser dans la carte d'assemblage qui fait appel à un autre composant d'assemblage *<(Composants Co1 et Co2), Assembler les composants Co1 et Co2 avec la stratégie d'association>* développé à la section 3.5 de ce chapitre. Dans cette section nous illustrons également l'assemblage des composants CREWS-*L'Ecritoire* et *Albert.* Le résultat de l'assemblage des modèles processus est résumé à la Figure 118.

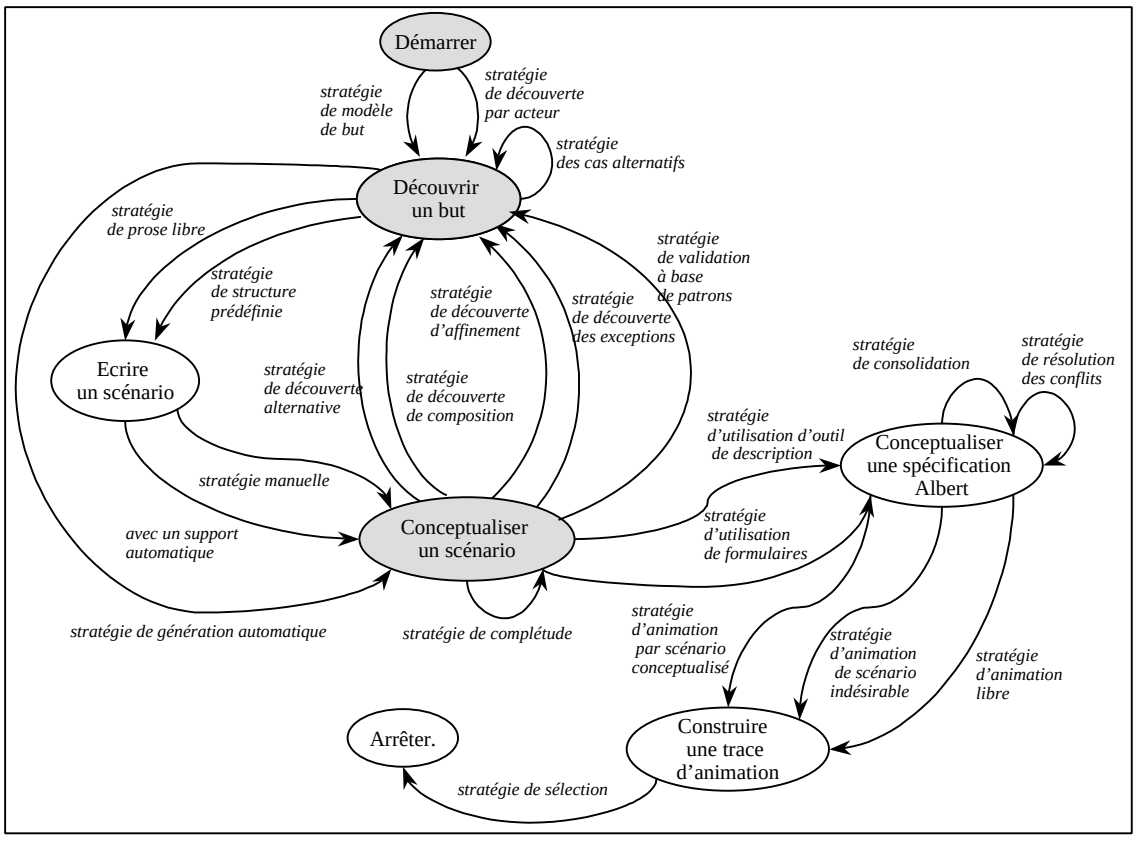


Figure 118 : Résultat d'assemblage des composants N° 1, 3, 9 et 10

Puisque la carte provisoire n'est toujours pas couverte par la carte de la méthode en construction, on poursuit la recherche d'autres composants que l'on assemble ensuite avec la méthode en construction de la même manière que les assemblages précédents. On termine le processus d'assemblage en suivant la stratégie de complétude qui aide à vérifier si la carte de la méthode en construction couvre complètement la carte des exigences et si les directives associées à cette carte sont assez riches pour satisfaire les besoins de l'ingénieur d'applications. Par ailleurs, cette méthode peut être complétée par la suite en appliquant d'autres composants d'assemblage.

La démarche appliquée est résumée à la Figure 119 qui visualise les sections sélectionnées lors de la construction de la nouvelle méthode ainsi que l'ordre dans lequel elles ont été sélectionnées.

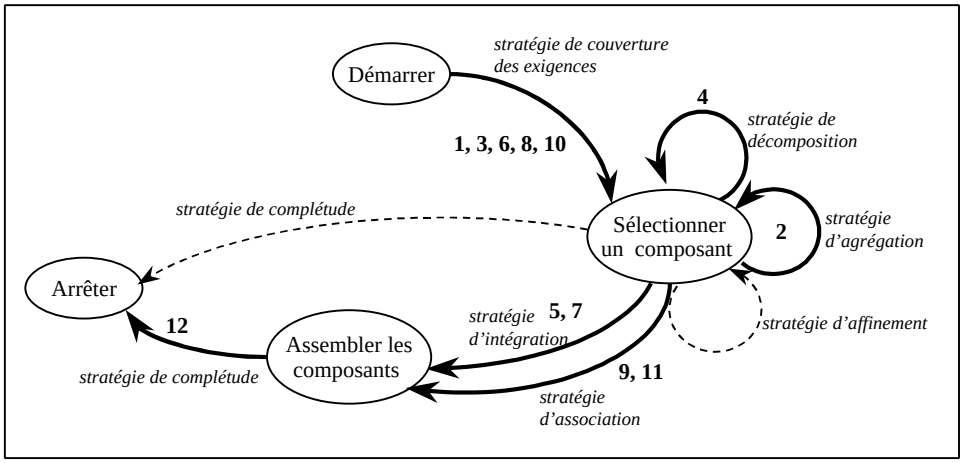


Figure 119 : La démarche utilisée pour la construction de la nouvelle méthode

3.2 Complétude d'un composant par une nouvelle démarche

3.2.1 Descripteur

Situation de réutilisation

Domaine d'application **:** Ingénierie de méthodes

Activité **:** Adaptation d'une méthode par assemblage de composants de méthode

Intention de réutilisation

Enrichir une méthode par une démarche complémentaire

Objectif

L'objectif de ce type d'assemblage est d'enrichir la directive d'un composant si celle-ci est trop pauvre en la complétant par celle d'un autre composant. Ce composant d'assemblage peut aussi résoudre le problème de manque de définition de démarches dans les méthodes existantes. Quand un composant de méthode a un modèle de produit bien défini, mais la démarche pour construire le produit

n'existe pas ou elle est trop informelle, nous proposons de lui adapter la démarche définie dans un autre composant. Les modèles de produit des deux composants doivent avoir des éléments similaires.

Type : Agrégat

3.2.2 Composant

Identifiant : CA4

Signature

***Situation* :** Composant de méthode Co1

***Intention* :** Construire un composant de méthode avec la stratégie de complétude

Directive

Type de directive : Stratégique

Représentation formelle : Le processus d'assemblage est représenté par une carte à la Figure 120. Les directives associées à la carte sont répertoriées dans le Tableau 3.

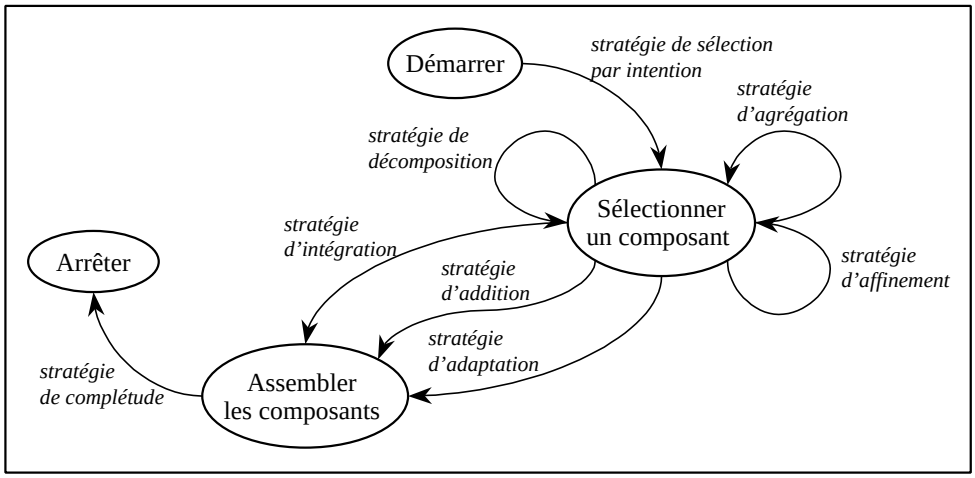
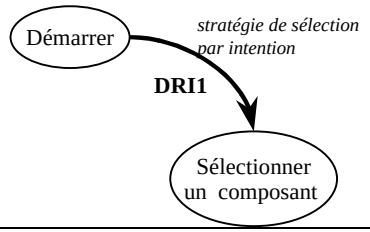
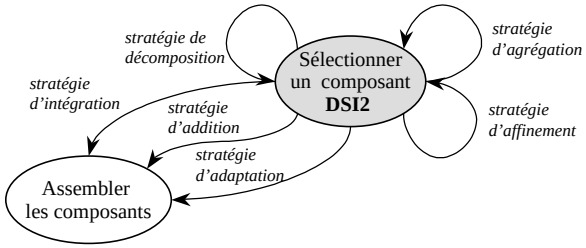
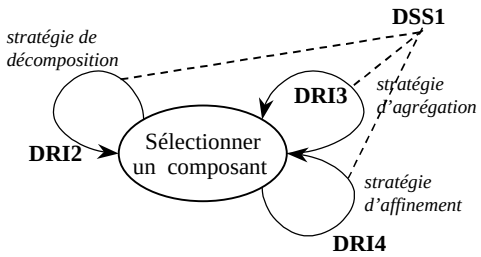
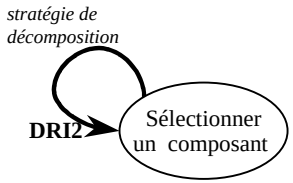
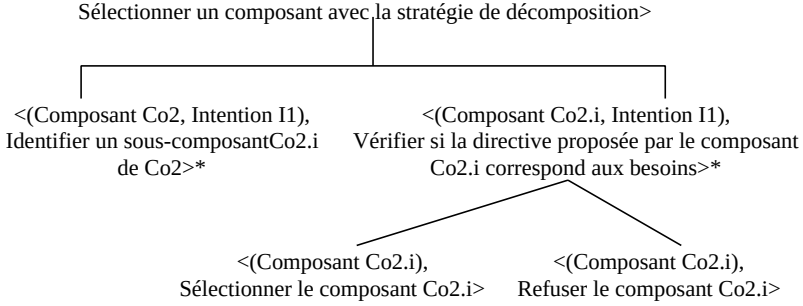
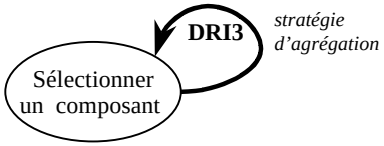
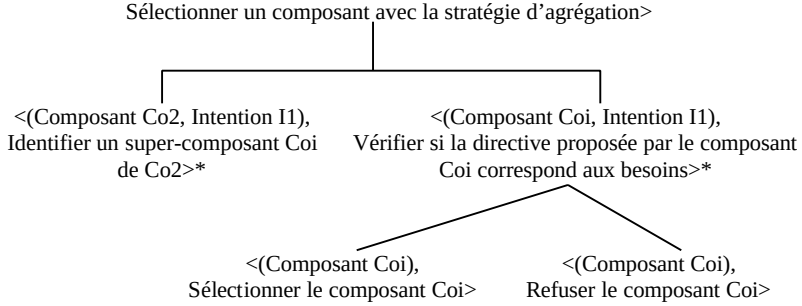
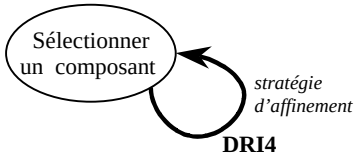
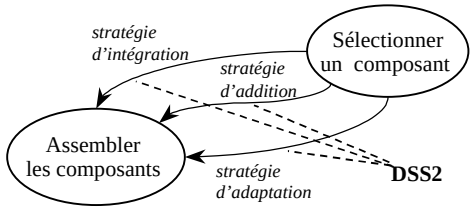
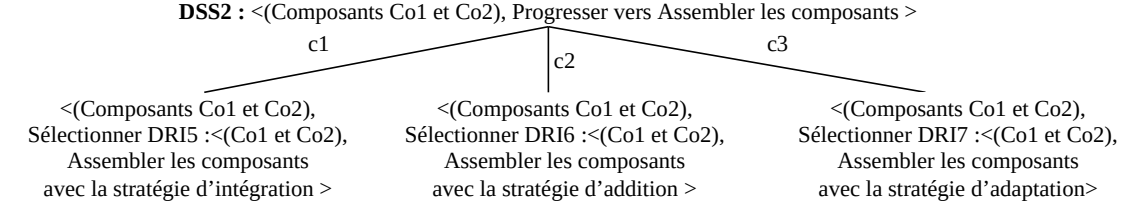
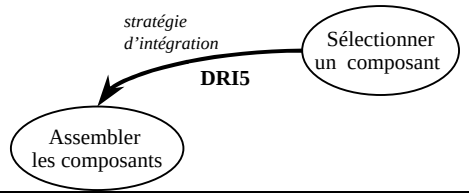
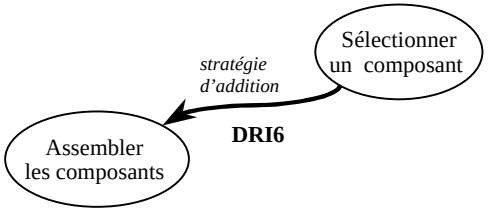
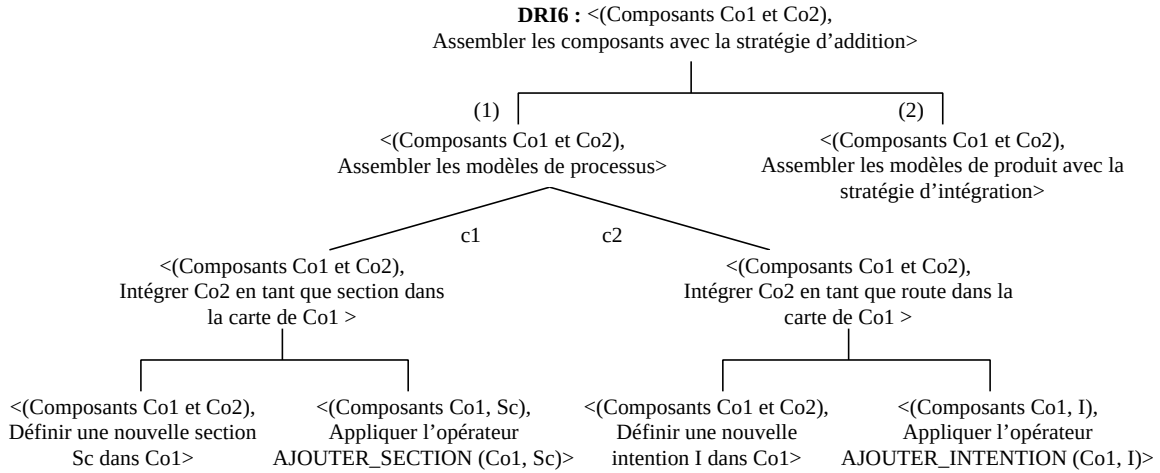


Figure 120 : Carte de processus d'assemblage pour enrichir la directive d'un composant

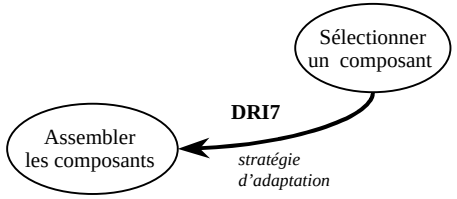
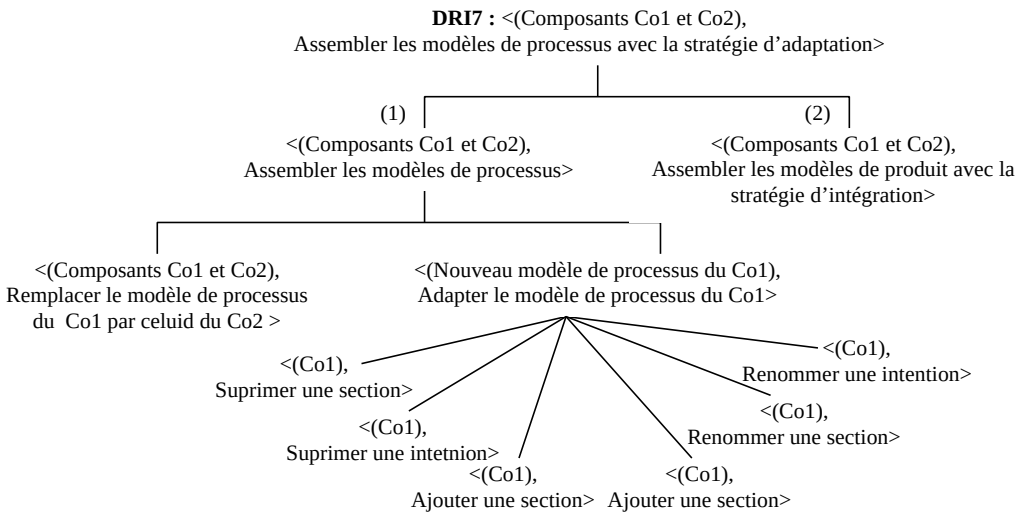
N°	Situation dans la carte de processus d'assemblage	Directive correspondant à la situation
1		**DRI1 :** <(Composant Co1, Intention I1), Sélectionner un composant Co2 avec la stratégie de sélection par intention> Une intention dans la carte du composant Co1 est identifiée au préalable par l'ingénieur d'applications pour laquelle il veut ajouter une nouvelle démarche de réalisation. 1. Sélectionner les composants ayant l'intention I1(ou une intention similaire) dans leur signature.
2		**DSI2 :** <(Composants Co1, Intention I1, Composant Co2), Progresser de Sélectionner un composant> c1 : <(Composants Co1 et Co2, Intention I1), Sélectionner DSS1 :<(Co1 et Co2, I1), Progresser vers Sélectionner un composant >> c2 : <(Composants Co1 et Co2), Sélectionner DSS2 :<(Co1 et Co2), Progresser vers Assembler les composants >> Critères de choix : c1 : le composant sélectionné ne correspond pas au résultat souhaité par l'ingénieur d'applications ; c2 : le composant sélectionné répond aux besoins de l'ingénieur d'applications.
3		**DSS1 :** <(Composants Co1 et Co2, Intention I1), Progresser vers Sélectionner un composant> c1 : <(Composants Co1 et Co2, Intention I1), Sélectionner DRI2 :<(I1, Co2), Sélectionner un composant avec la stratégie de décomposition >> c2 : <(Composants Co1 et Co2, Intention I1), Sélectionner DRI3 :<(I1, Co2), Sélectionner un composant avec la stratégie d'agrégation >> c3 : <(Composants Co1, Intention I1), Sélectionner DRI4 :<(Co1, I1), Sélectionner un composant avec la stratégie d'affinement>> Critères de choix : c1 : le composant Co2 est de type agrégat et sa directive est plus riche que ne le demande le composant Co1 ; c2 : le composant Co2 fait partie d'au moins un composant agrégat ; c3 : la directive du composant Co2 n'est pas assez riche.

<table>
<tr>
<td>4</td>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>5</td>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>6</td>
<td> 
</td>
<td>DRI4 : <(Composant Co1, I1), Sélectionner un composant avec la stratégie d’affinement>
Affiner la recherche d’un composant en remplaçant les paramètres de recherche par des synonymes (le verbe de l’intention, la cible).</td>
</tr>
<tr>
<td>7</td>
<td> 
</td>
<td> 

Critères de choix :</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td>c1 : les deux composants ont des intentions similaires ; le composant Co2 couvre plus d’une section du composant Co1;
c2 : le composant Co2 peut être ajouté en tant que section ou route dans la carte du composant Co1 ;
c3 : le composant Co2 n’a pas de directive ou elle est mal définie.</td></tr>
<tr><td>8</td><td> 
</td><td>DRI5 : <(Composants Co1 et Co2), Assembler les composants avec la stratégie d’intégration>
Appliquer le composant d’assemblage CA3-1 (Section 3.4 de ce chapitre).</td></tr>
<tr><td>9</td><td> 
</td><td> 

(1) Définir la directive du composant Co2 en tant que section (c1) ou route de sections (c2) dans la carte du composant Co2.
(2) Appliquer le processus d’assemblage par intégration : La partie concernant l’assemblage des modèles de produit dans le composant d’assemblage CA3-1 (Section 3.4 de ce chapitre).</td></tr>
</table>

<table>
<tr>
<td>10</td>
<td> 
</td>
<td> 

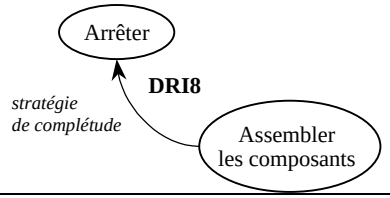
(1) Remplacer la directive du composant Co1 par celle du composant Co2.
(2) Appliquer le processus d'assemblage par intégration : La partie concernant l'assemblage des modèles de produit dans le composant d'assemblage CA3-1 (Section 3.4 de ce chapitre).</td>
</tr>
<tr>
<td>11</td>
<td> 
</td>
<td>DRI8 : <(Composant assemblé), Arrêter le processus d'assemblage avec la stratégie de complétude>
Appliquer les règles de complétude et de cohérence sur les produit et processus assemblés.</td>
</tr>
</table>

Tableau 3 : Directives associées à la carte d'assemblage

Description

Situation 1 : L'ingénieur d'applications est invité à démarrer le processus d'assemblage par la sélection du composant de méthode dont la directive pourrait enrichir la directive du composant initial. La sélection d'un tel composant est basée sur l'identification d'une intention dans la carte du composant initial dont la réalisation n'est pas assurée d'une manière satisfaisante. Par conséquent, les paramètres de la recherche sont basés sur le verbe et la cible de cette intention.

Situation 2 : Une fois qu'un composant est sélectionné, la DSI2 propose deux possibilités :

1. rester sur la même intention *"Sélectionner un composant"* et affiner la sélection d'un composant en le décomposant (s'il est de type agrégat) ou en cherchant des agrégats dont celui-ci fait partie, ou même en redéfinissant les paramètres de la recherche ou
2. progresser vers l'assemblage des composants si la sélection du composant a été satisfaisante.

Situation 3 : Admettons que l'ingénieur d'applications décide d'affiner la sélection d'un composant en restant sur l'intention *"Sélectionner un composant"*. La directive de sélection de stratégies DSI1 associée à cette intention lui propose de choisir une parmi trois stratégies possibles : la *stratégie de décomposition*, la *stratégie d'agrégation* et la *stratégie d'affinement*.

Situation 4 : Si le composant sélectionné est de type agrégat et la directive qu'il propose est trop importante par rapport à celle demandée par le composant initial, la DRI2 associée à la section *<Sélectionner un composant, Sélectionner un composant, Stratégie de décomposition>* aide à décomposer ce composant et à sélectionner un de ses sous-composants ayant l'intention plus proche de l'intention demandée que le composant de départ. L'ingénieur de développement doit vérifier ensuite, si la directive proposée par ce composant répond mieux à ses besoins que la directive du composant du départ.

Situation 5 : Si au contraire, la directive du composant sélectionné ne répond pas complètement à la demande du composant du départ et que ce composant fait partie d'au moins un composant agrégat, la DRI3 associée à la section *<Sélectionner un composant, Sélectionner un composant, Stratégie d'agrégation>* aide à sélectionner un de ses super-composants et à vérifier si le processus qu'il propose est plus riche que celui proposé par le composant du départ.

Situation 6 : Finalement, si la directive du composant ne correspond pas à la démarche souhaitée par le composant du départ, la DRI4 associée à la section *<Sélectionner un composant, Sélectionner un composant, Stratégie d'affinement>* propose de recommencer les recherches en les affinant, en proposant des valeurs synonymes du verbe et de la cible de l'intention qui sert de base aux recherches.

Situation 7 : Une fois que la sélection d'un composant est validée l'ingénieur d'applications progresse vers l'assemblage du composant initial avec le composant sélectionné. La directive de progression DSS2 propose trois stratégies d'assemblage : la stratégie *d'intégration*, la stratégie *d'addition* et la stratégie *d'adaptation*. L'ingénieur d'applications choisit la première stratégie si la carte du composant sélectionné comporte des intentions communes avec la carte du composant du départ. La deuxième stratégie peut être choisie si la directive du composant sélectionné correspond à une section ou une

route dans la carte du composant du départ. Finalement, la troisième stratégie propose de remplacer la directive du composant de départ par la directive du composant sélectionné en l'adaptant si nécessaire. Dans tous les trois cas, le processus d'assemblage des modèles de produit correspond à celui proposé par le composant d'assemblage CA3-1, c'est-à-dire à l'assemblage par intégration (Section 3.4).

Situation 8 : La stratégie *d'intégration* pour assembler des composants correspond au processus proposé par le composant d'assemblage CA3-1 (Section 3.4). La DRI5 associée à la section *<Sélectionner un composant de méthode, Assembler les modèles de processus, Stratégie d'intégration>* est donc représentée par le composant CA3-1.

Situation 9 : La stratégie *d'addition* d'assemblage des composants concerne le cas où la directive du composant sélectionné n'est pas présentée par une carte mais par une directive tactique ou même une directive simple. Elle représente en général une section dans la carte du composant de départ. Si c'est une directive tactique de type plan, elle peut être transformée en une route de sections en ajoutant des intentions intermédiaires.

Puisque les modèles de produit des deux composants ont des éléments similaires, leur assemblage est toujours de type *par fusion des éléments communs*. La partie concernant l'assemblage des modèles de produit dans la DRI7 réutilise la partie concernant l'assemblage des modèles de produit appliqué par le composant CA3-1 (Section 3.4). Nous réutilisons cette partie afin d'éviter de la redéfinir à nouveau.

Situation 10 : La stratégie *d'adaptation* pour assembler des composants consiste à remplacer le modèle de processus du composant du départ par celui du composant sélectionné. Ce processus est assuré par la DRI7. On peut avoir la nécessité d'adapter ce modèle de processus en fonction du modèle de produit en renommant certaines intentions et/ou sections, en supprimant certaines sections ou même des intentions ou bien en ajoutant de nouvelles sections et/ou des nouvelles intentions. Comme la DRI6, la DRI7 applique l'assemblage par intégration sur les modèles de produit des composants.

Situation 11 : L'arrêt de processus d'assemblage des modèles de produit et de processus est guidé par la DRI8 qui aide à appliquer les règles de complétude sur le composant obtenu..

3.2.3 Exemple d'application du composant d'assemblage

Prenons à nouveau comme exemple le composant de méthode OOSE présenté à la Figure 128 et à la Figure 130. Comme nous l'avons vu à la section 3.4.3, le composant OOSE propose une démarche pour conceptualiser des besoins d'un système d'information sous forme des *cas d'utilisation*. Le point faible de cette démarche est le guidage dans l'écriture des scénarios composant les cas d'utilisation. Ce guidage est pratiquement inexistant. On construit une requête dont l'argument de recherche est l'intention de la signature de composant. On propose plusieurs valeurs synonymes pour le verbe et la cible de l'intention. Par exemple, on cherche des composants dont l'intention est *"Conceptualiser un cas d'utilisation"* ou *"Conceptualiser un scénario"* ou *"Ecrire un scénario"*. Supposons que la liste des signatures des composants correspondant à la requête soit la suivante :

1. <(But), Ecrire un scénario en prose libre>
2. <(But), Ecrire un scénario avec une structure prédéfinie>

3. <(But), Ecrire un scénario avec la stratégie linguistique de CREWS-*L'Ecritoire*>
4. <(Scénario), Conceptualiser un scénario avec un support automatique>
5. <(Scénario), Conceptualiser un scénario avec la stratégie manuelle>
6. <(Scénario), Conceptualiser un scénario avec l'outil *L'Ecritoire*>
7. <(But), Conceptualiser un scénario avec la stratégie de *CEWS-L'Ecritoire*>

Tous les composants sélectionnés sont des éléments du composant agrégat CREWS-*L'Ecritoire*. Comme le montre la carte de ce dernier présentée à la Figure 131, le composant N°3 est aussi un agrégat comportant les deux premiers composants associés par une directive de type *parallélisme*. De la même manière le composant N°6 est un agrégat des composants N°4 et 5. L'enchaînement des composants N°3 et N°6 est également un composant de type agrégat ayant dans cette liste le numéro 7. Après avoir analysé tous ces composants on décide de sélectionner le dernier, celui qui comporte l'écriture et la conceptualisation des scénarios.

On passe ensuite à l'assemblage des modèles de processus du composant OOSE et du composant sélectionné. On choisit la stratégie *d'addition* (Figure 120) car la directive du composant sélectionné peut être ajoutée dans la carte de OOSE comme une section supplémentaire entre les intentions *"Découvrir un cas d'utilisation"* et *"Conceptualiser un cas d'utilisation"*. Mais avant tout il est nécessaire de l'adapter afin qu'elle permette d'obtenir des cas d'utilisation à la place des scénarios indépendants ; un cas d'utilisation étant un ensemble de scénarios alternatifs. Par conséquent, on doit compléter la démarche du composant sélectionné par l'intégration des scénarios alternatifs en un cas d'utilisation. Le résultat de cet assemblage est illustré à la Figure 121 (a).

Il y a aussi une autre possibilité d'assemblage de ces composants. Puisque le composant sélectionné est un enchaînement de plusieurs sous-composants et qu'il représente une route de sections dans la carte de CREWS-*L'Ecritoire*, il peut être décomposé et ajouté à la carte de OOSE comme une nouvelle route. Cet assemblage nécessite d'ajouter deux intentions *"Ecrire un scénario"* et *"Conceptualiser un scénario"* dans la carte de OOSE ainsi que quatre sections correspondant chacune à un de ces sous-composants. Le résultat d'un tel assemblage est plus flexible que le précédent. Il est illustré à la Figure 121 (b).

Dans les deux cas, l'ancienne section <*Découvrir un cas d'utilisation, Conceptualiser un cas d'utilisation, Stratégie de cas normal d'abord*> (Figure 130) est éliminée de la carte car elle est remplacée par celle de CREWS-*L'Ecritoire*.

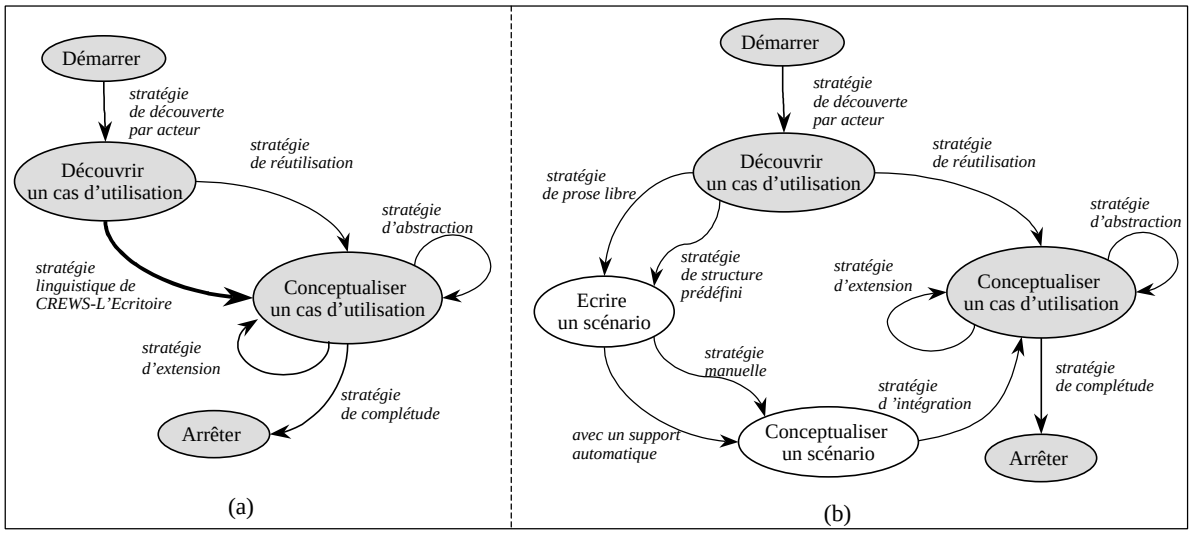


Figure 121 : Résultat d'assemblage des modèles de processus

L'assemblage des modèles de produit consiste à remplacer le concept *scénario* de OOSE (Figure 128) par celui de CREWS-*L'Ecritoire* (Figure 129), car la structure de ce dernier est beaucoup plus complexe et à fusionner le concept *acteur* de OOSE et le concept *agent* de CREWS-*L'Ecritoire*. Le résultat est représenté à la Figure 122.

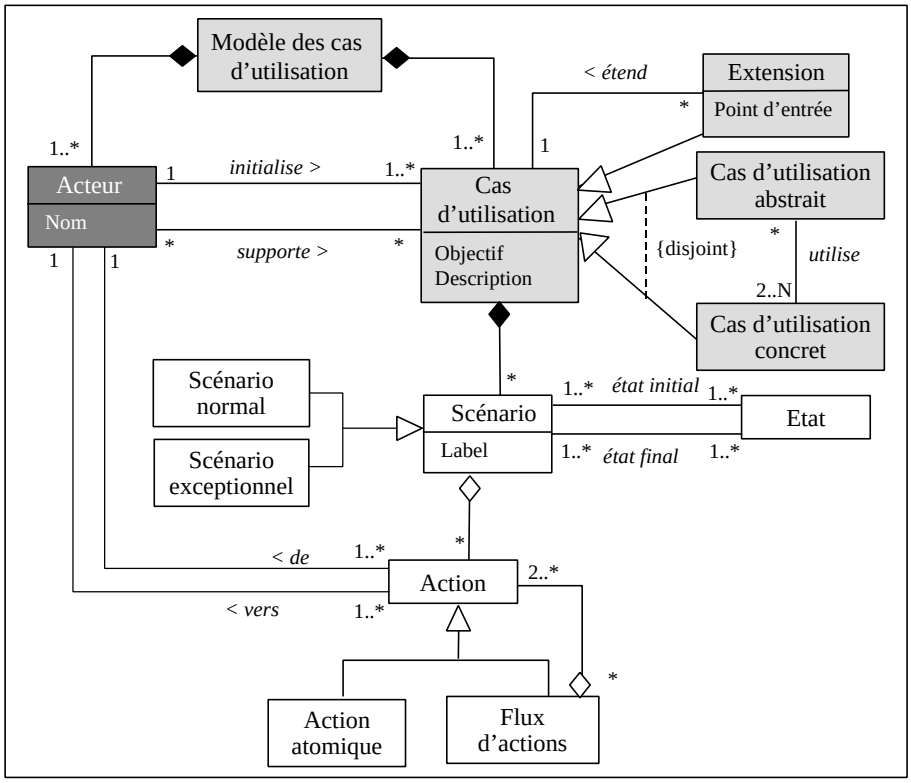


Figure 122 : Résultat d'assemblage des modèles de produit

La démarche appliquée lors de l'assemblage pour enrichir un composant est résumée à la Figure 123 qui précise les sections sélectionnées dans la carte de processus d'assemblage ainsi que l'ordre dans lequel elles ont été choisies.

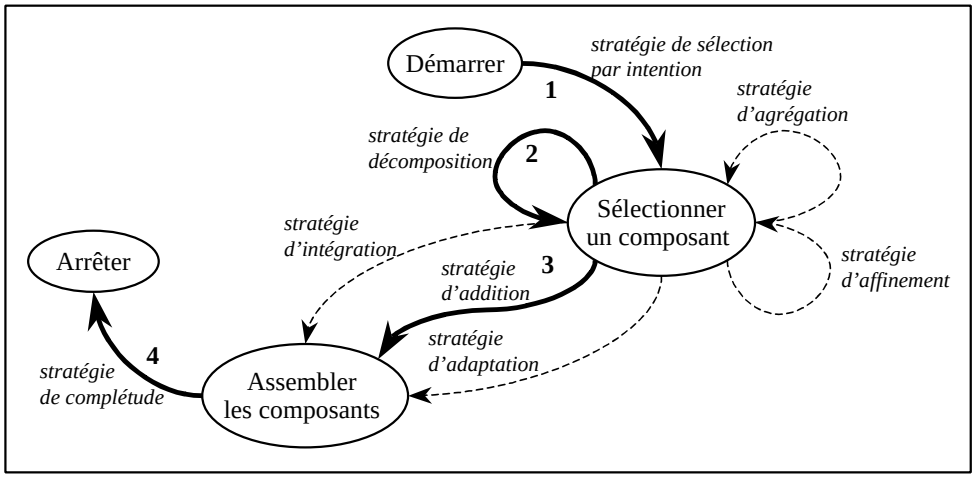


Figure 123 : La démarche utilisée pour enrichir la directive du OOSE

3.3 Enrichissement d'un composant par une nouvelle fonctionnalité

3.3.1 Descripteur

Situation de réutilisation

Domaine d'application **:** Ingénierie de méthodes

Activité **:** Adaptation d'une méthode par assemblage de composants de méthode

Intention de réutilisation

Etendre une méthode par une fonctionnalité complémentaire

Objectif

L'objectif de ce type d'assemblage est d'enrichir un composant par une nouvelle fonctionnalité (un nouveau modèle) complémentaire à celle du composant. Cette fonctionnalité doit servir à construire un nouveau produit à base de celui du composant initial ou à modifier le produit du composant initial. Ce composant d'assemblage permet d'enrichir une méthode existante par un nouveau modèle sans modifications importantes sur la méthode. Les modèles de produit des deux composants ne nécessitent pas d'avoir des similarités.

Type : Agrégat

3.3.2 Composant

Identifiant : CA5

Signature

Situation **:** Composant de méthode Co1

***Intention* :** Construire un composant de méthode avec la stratégie d'extension

Directive

Type de directive : Stratégique

Représentation formelle : Le processus d'assemblage est représenté par une carte à la Figure 124. Les directives associées à la carte sont répertoriées dans le Tableau 3.

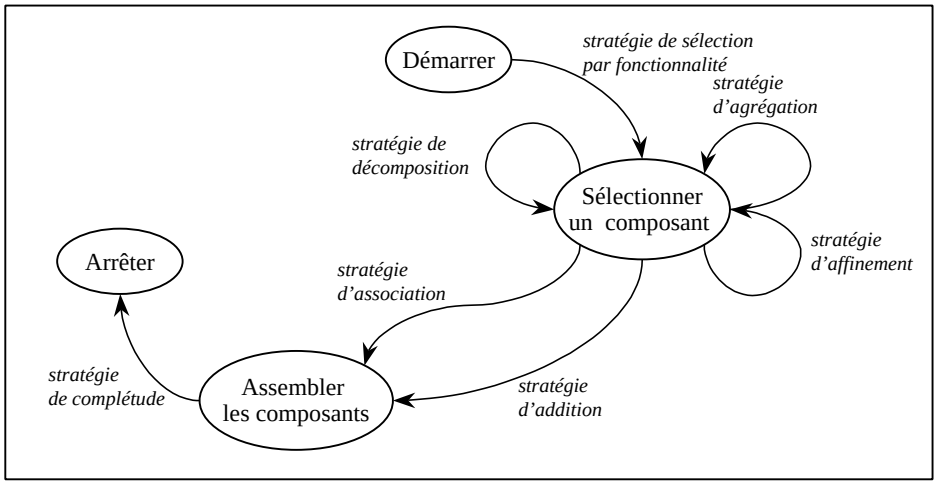
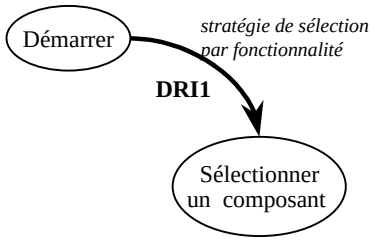
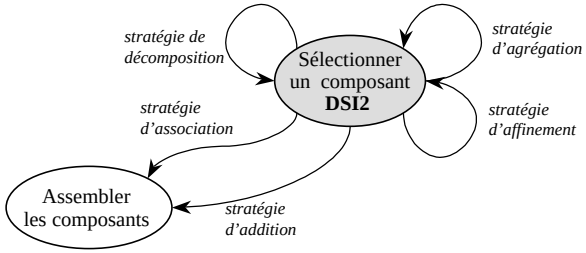
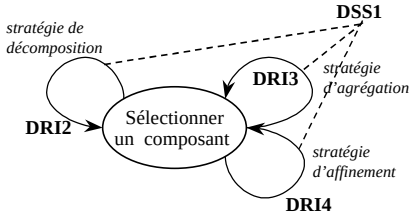
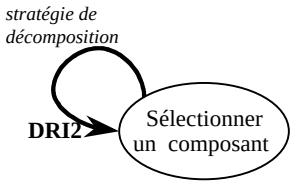
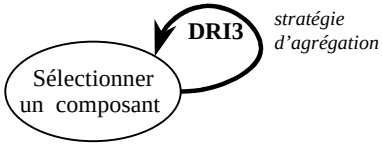
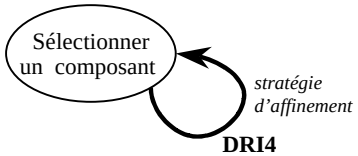
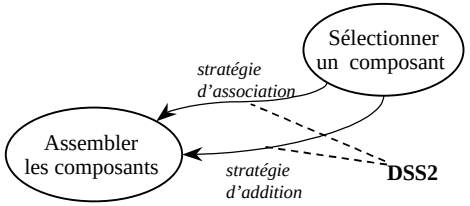
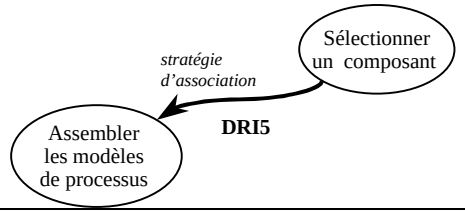
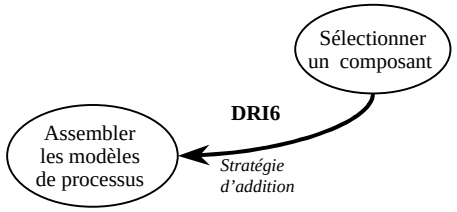


Figure 124 : Carte de processus d'assemblage par extension

Nº	Situation dans la carte de processus d'assemblage	Directive correspondant à la situation
1		**DRI1 :** <(Composant Co1, Intention I1), Sélectionner un composant Co2 avec la stratégie de sélection par fonctionnalité complémentaire> La fonctionnalité que l'on veut ajouter au composant Co1 doit être formulée au préalable en terme d'une intention en appliquant le composant d'assemblage CA2. Sélectionner les composants ayant cette intention (ou une intention similaire) dans leur signature.
2		**DSI2 :** <(Composants Co1 et Co2, Intention I1), Progresser de Sélectionner un composant> c1 : <(Composants Co1 et Co2, Intention I1), Sélectionner DSS1 :<(Co1 et Co2, I1), Progresser vers Sélectionner un composant >> c2 : <(Composants Co1 et Co2), Sélectionner DSS2 :<(Co1 et Co2), Progresser vers Assembler les composants >> Critères de choix : a1 : le composant sélectionné ne correspond pas au résultat souhaité par l'ingénieur d'applications ; a2 : le composant sélectionné répond aux besoins de l'ingénieur d'applications.
3		**DSS1 :** <(Composants Co1 et Co2, Intention I1), Progresser vers Sélectionner un composant> c1 : <(Composants Co1 et Co2, Intention I1), Sélectionner DRI2 :<(I1, Co2), Sélectionner un composant avec la stratégie de décomposition >> c2 : <(Composants Co1 et Co2, Intention I1), Sélectionner DRI3 :<(I1, Co2), Sélectionner un composant avec la stratégie d'agrégation >> c3 : <(Composants Co1, Intention I1), Sélectionner DRI4 :<(Co1, I1), Sélectionner un composant avec la stratégie d'affinement>> c1 : Le composant sélectionné propose une fonctionnalité de conception plus importante que l'ingénieur d'applications n'en ait besoin ; c2 : Le composant sélectionné propose une fonctionnalité de conception qui n'a pas toutes les qualités requises par l'ingénieur d'applications ;

		c3 : : Le composant sélectionné propose une fonctionnalité de conception qui ne correspond pas aux besoins de l'ingénieur d'applications.
4		**DRI2** :<(Composant Co2, Intention I1), Sélectionner un composant avec la stratégie de décomposition> <(Composant Co2, Intention I1), Identifier un sous-composantCo2.i de Co2 ayant comme intention> <(Composant Co2.i), Valider le produit proposé par le composant Co2.i>* <(Composant Co2.i), Sélectionner le composant Co2.i> <(Composant Co2.i), Refuser le composant Co2.i>
5		**DRI2** :<(Intentions I1, Composant Co2), Sélectionner un composant avec la stratégie d'agrégation> <(Composant Co2, Intention I1), Identifier un super-composant Coi de Co2>* <(Composant Coi), Valider le produit proposé par le composant Coi>* <(Composant Coi), Sélectionner le composant Coi> <(Composant Coi), Refuser le composant Coi>
6		**DRI4** : <(Composant Co1, I1), Sélectionner un composant avec la stratégie d'affinement> Affiner la recherche d'un composant en remplaçant les paramètres de recherche par des synonymes (le verbe de l'intention, la cible).

<table>
<tr>
<td>7</td>
<td> 
</td>
<td>DSS2 : <(Composants Co1 et Co2), Progresser vers Assembler les modèles de processus>
c1 : <(Composants Co1 et Co2), Sélectionner DRI5 :<(Co1 et Co2), Assembler les composants avec la stratégie d'association>
c2 : <(Composants Co1 et Co2), Sélectionner DRI6 :<(Co1 et Co2), Assembler les composants avec la stratégie d'addition>
Critères de choix :
c1 : les deux composants n'ont pas d'intentions similaires ;
c2 : le composant Co2 peut être ajouté en tant que section ou route dans la carte du composant Co1.</td>
</tr>
<tr>
<td>8</td>
<td> 
</td>
<td>DRI5 : <(Composants Co1 et Co2), Assembler les modèles de processus avec la stratégie d'association>
Appliquer le processus d'assemblage par association : la partie concernant l'assemblage des modèles de processus dans le composant d'assemblage CA3-2 (Section 3.5 de ce chapitre).</td>
</tr>
<tr>
<td>9</td>
<td> 
</td>
<td>DRI6 : <(Composants Co1 et Co2), Assembler les composants avec la stratégie d'addition>
(1) <(Composants Co1 et Co2), Assembler les modèles de processus avec la stratégie d'addition>
(2) <(Composants Co1 et Co2), Assembler les modèles de produit avec la stratégie d'association>
(1) <(Composants Co1 et Co2), Identifier une intention I à base de la signature du composant Co2>
(2) <(Composants Co1, I), Appliquer l'opérateur AJOUTER_INTENTION (Co1, I)>
(1) Définir la directive du composant Co2 en tant que nouvelle intention dans la carte de Co1.
(2) Appliquer le processus d'assemblage par association : La partie concernant l'assemblage des modèles de produit dans le composant d'assemblage CA3-2 (Section 3.5 de ce chapitre).</td>
</tr>
</table>

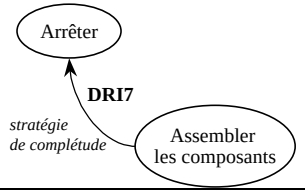
10		**DRI7** : <(Composant assemblé), Arrêter le processus d'assemblage avec la stratégie de complétude> Appliquer les règles de complétude et de cohérence sur le produit et le processus assemblés.

Tableau 4 : Directives associées aux processus d'assemblage par extension

Description

Situation 1 : L'ingénieur d'applications est invité à démarrer le processus d'assemblage par la sélection du composant de méthode qui pourrait enrichir le composant initial avec une nouvelle fonctionnalité. Pour construire la requête de sélection d'un tel composant, il est nécessaire que la fonctionnalité à ajouter soit formulée en terme d'une intention d'ingénierie, en sachant que toute intention doit être exprimée par un verbe et des paramètres. Le paramètre cible est obligatoire tandis que les autres paramètres sont optionnels mais ils permettent de mieux cerner le composant le mieux adapté à la situation.

Situation 2 : Après avoir sélectionné un composant l'ingénieur d'applications est guidé par la DSI2 qui est identique à la directive correspondante du composant CA4 (voir la situation 2 à la section 3.2.2).

Situation 3 : L'affinement de la sélection d'un composant le mieux adapté à la demande est guidé par la DSS1 qui propose trois stratégies de sélection : la *stratégie de décomposition*, la *stratégie d'agrégation* et la *stratégie d'affinement* détaillées par la suite.

Si le composant sélectionné auparavant propose une fonctionnalité de conception plus importante que l'ingénieur d'applications n'en ait besoin (le produit proposé est trop complexe) et si ce composant est un agrégat, l'ingénieur d'applications est invité à choisir la *stratégie de décomposition* qui est détaillée à la situation 4 du Tableau 4.

Si au contraire, la fonctionnalité proposée par le composant sélectionné n'est pas assez puissante et qu'elle n'a pas toutes les qualités requises par l'ingénieur d'applications, la DSS suggère de choisir la *stratégie d'agrégation* qui est présentée à la situation 5 du Tableau 4.

Finalement, si le produit proposé par le composant sélectionné ne correspond pas aux besoins de l'ingénieur d'applications, il peut choisir la *stratégie d'affinement* décrite à la situation 6 du Tableau 4.

Situation 4 : La DRI2 associée à la section <*Sélectionner un composant, Sélectionner un composant, Stratégie de décomposition*> propose de décomposer le composant agrégat sélectionné au préalable et d'analyser tous ses sous-composants afin de retrouver celui qui satisfasse au mieux les besoins de l'ingénieur d'applications.

Situation 5 : La DRI3, associée à la section <*Sélectionner un composant, Sélectionner un composant, Stratégie d'agrégation*>, propose de trouver des composants agrégats comportant le composant sélectionné et de vérifier si la fonctionnalité qu'ils proposent n'est pas mieux adaptée aux besoins de l'ingénieur d'applications. C'est le modèle de produit du composant qui est validé.

Situation 6 : La DRI5 associée à la section <*Sélectionner un composant, Sélectionner un composant, Stratégie d'affinement*> propose d'affiner la requête de recherche en remplaçant les valeurs des paramètres (verbe, cible,..) par des synonymes.

Situation 7 : Quand l'ingénieur d'applications décide d'assembler le composant initial et le composant qu'il vient d'extraire de la base de composants, la DSS2 lui propose deux solutions :

1) appliquer la *stratégie d'association* si le modèle de processus du composant sélectionné est de type stratégique (situation 5) ou

2) appliquer la *stratégie d'addition* si le processus du composant sélectionné est de type tactique ou simple (situation 6).

Situation 8 : La DRI5 associée à la section <*Sélectionner un composant, Assembler les composants, Stratégie d'association*> applique le composant d'assemblage CA3-2 (Section 3.5).

Situation 9 : La stratégie *d'addition* d'assemblage de composants propose d'introduire le nouveau composant en tant que nouvelle intention dans la carte du composant initial. La DRI6 associée à cette section consiste à représenter la directive du composant sélectionné par une intention et une ou plusieurs sections connectant cette intention avec la carte du composant initial.

Les modèles de produit des deux composants n'ont pas d'éléments communs. L'assemblage par association est donc appliqué pour assembler les deux modèles de produit. La DRI7 réutilise la partie concernant l'assemblage des modèles de produit appliqué par le composant CA3-2 (Section 3.5).

Situations 10 : L'arrêt de processus d'assemblage des modèles de produit et de processus consiste à valider les modèles obtenus ainsi que la cohérence entre les deux modèles en appliquant les règles de complétude et de cohérence présentées au Chapitre 5.

3.3.3 Exemple d'application

Pour illustrer l'application de ce composant de méthode on utilise de nouveau le composant de méthode OOSE dont les modèles de produit et de processus sont présentés à la Figure 128 et à la Figure 130 respectivement. Supposons que l'on décide de compléter ce composant par une fonctionnalité de documentation des cas d'utilisation obtenus en appliquant le composant OOSE. Pour extraire un composant correspondant de la base de méthodes on exprime cette fonctionnalité en terme d'intention *"Documenter un cas d'utilisation"*.

Puisque le processus de sélection de composant est similaire à celui utilisé par le composant CA4 *(Section 3.2)*, passons directement à l'assemblage des composants en supposant que l'on ait choisit le composant permettant de documenter les cas d'utilisation proposé par *Rational Software* dans le cadre du processus unifié de développement des logiciels [Schneider 98]. Ce composant propose une structure de description composée d'un ensemble de sections à remplir. Chaque section comporte une explication détaillée sur le type d'information qu'elle doit contenir. La Figure 125 visualise un extrait de cette structure.

Nom de cas d'utilisation
<Une brève description.>

Acteurs
<Une liste d'acteurs qui interagissent avec le cas d'utilisation.>

Priorité
<Quelle est l'importance de ce cas d'utilisation dans le projet.>

Pré conditions
<Une liste de conditions qui doivent être vraies avant que le cas d'utilisation ne démarre.>
…

Figure 125 : Extrait de la structure de description d'un cas d'utilisation

La directive de ce composant est informelle car elle propose tout simplement de remplir la structure de description en suivant les explications proposées. L'assemblage de ce composant avec le composant OOSE consiste à ajouter une nouvelle intention dans la carte du composant OOSE et un nouveau concept *Description* ayant comme attribut toutes les sections de la structure proposée dans le modèle de produit de OOSE. Le résultat d'assemblage est illustré à la Figure 126.

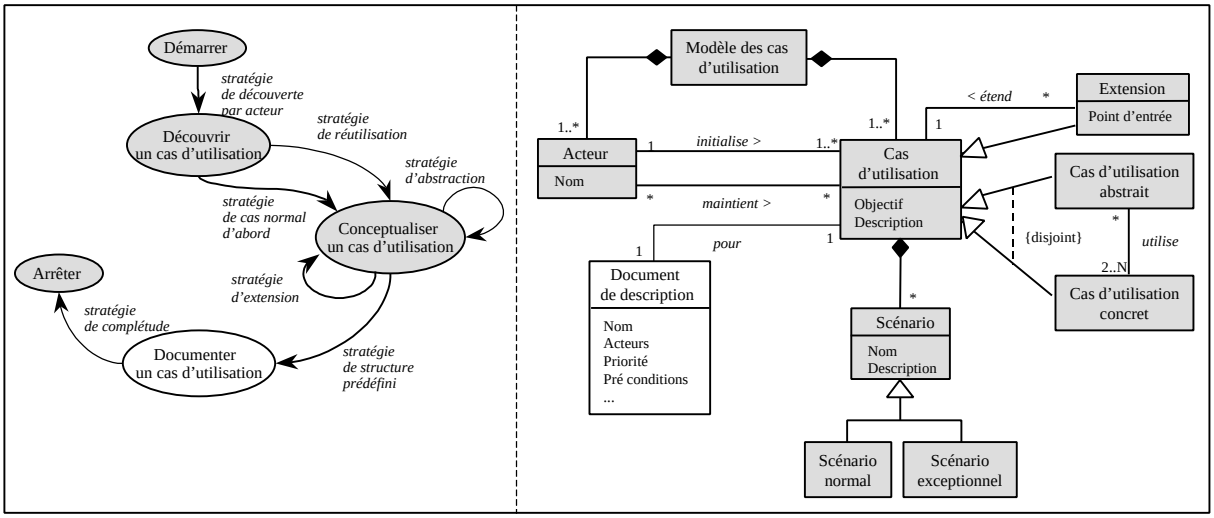


Figure 126 : Résultat d'assemblage

3.4 Assemblage des composants par intégration

3.4.1 Descripteur

Situation de réutilisation

Domaine d'application : Ingénierie de méthodes

Activité : Assemblage des composants de méthode

Intention de réutilisation

Intégrer des composants de méthode

Objectif

Ce composant d'assemblage propose d'intégrer deux composants de méthode équivalents en termes de résultat produit mais complémentaires en termes de la démarche permettant de développer le résultat. L'objectif d'assemblage de tels composants est d'obtenir un nouveau composant plus riche, plus

complet et plus flexible que les deux précédents en cumulant les points positifs de chaque composant et en éliminant leurs points négatifs. Le mode par intégration est utilisé pour assembler les composants.

Les composants de méthode à assembler doivent obligatoirement avoir des éléments communs dans leurs directives et dans leurs modèles de produit.

Type : Atomique

3.4.2 Composant

Identifiant : CA3-1

Signature

Situation : Deux composants de méthode Co1 et Co2

Intention : Assembler deux composants de méthode avec la stratégie d'intégration

Directive

Type de directive : Stratégique

Représentation formelle : Le processus d'assemblage est présenté par une carte à la Figure 127. Les directives associées à la carte sont répertoriées dans le Tableau 5.

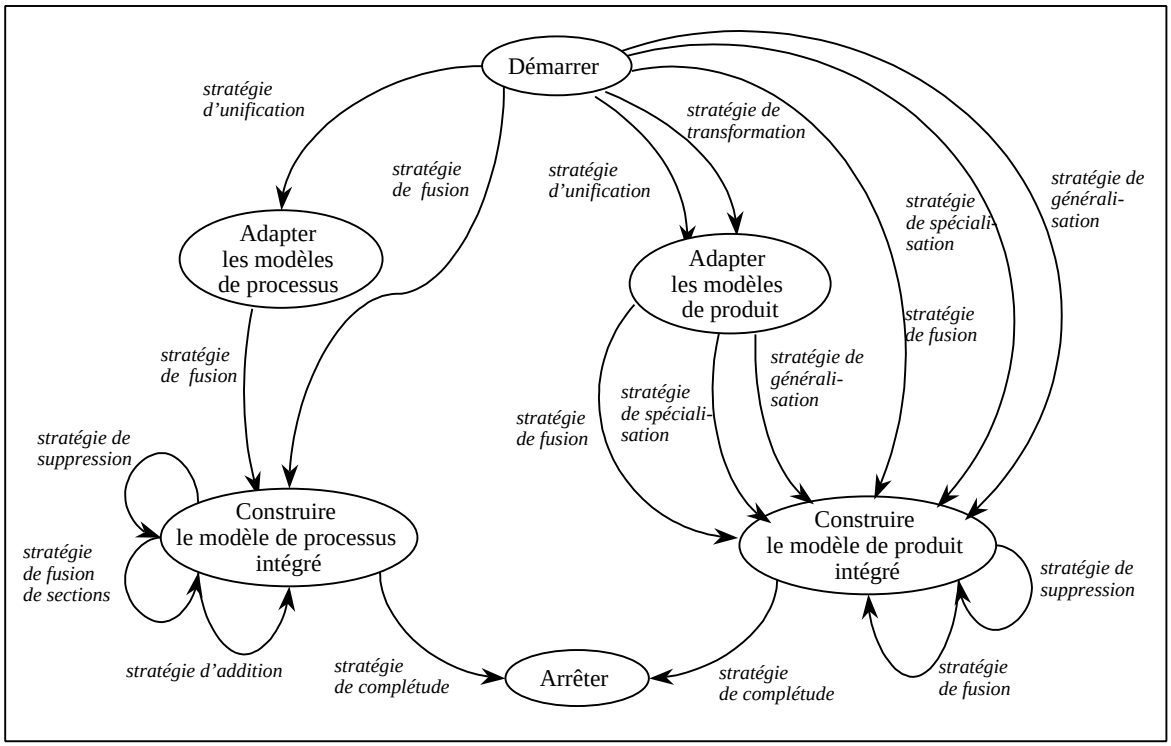
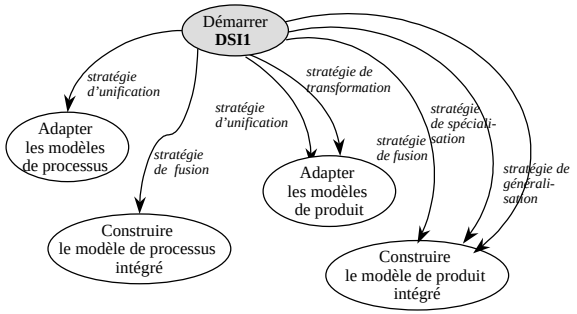
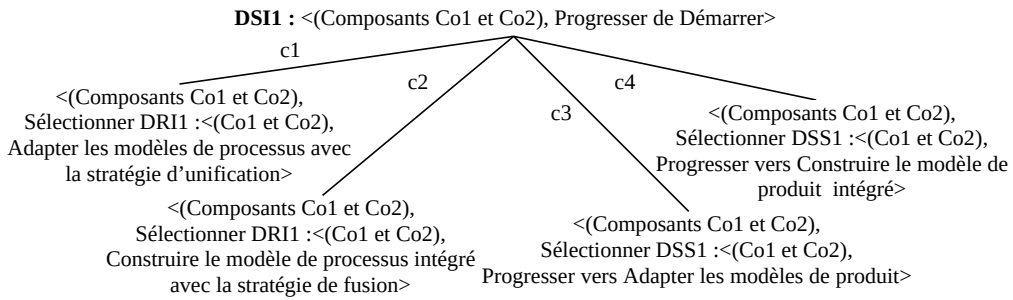
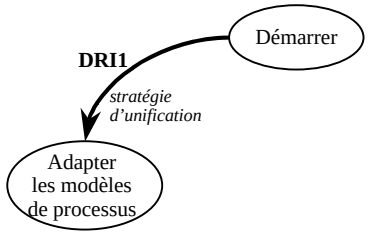
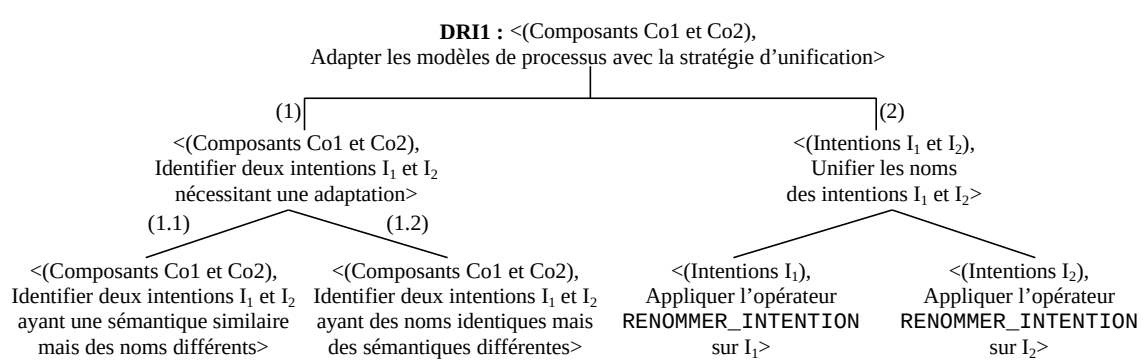


Figure 127 : Carte de processus d'assemblage de composants par intégration

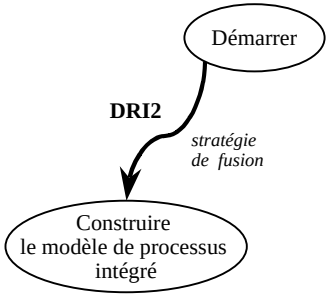
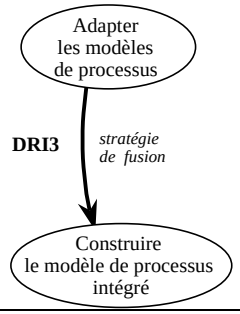
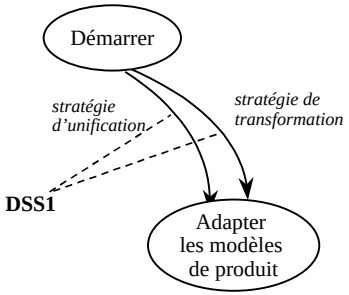
<table>
<tr><th>N°</th><th>Situation dans la carte de processus d'assemblage</th><th>Directive correspondant à la situation</th></tr>
<tr><td>1</td><td> 

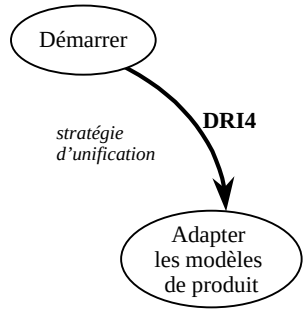
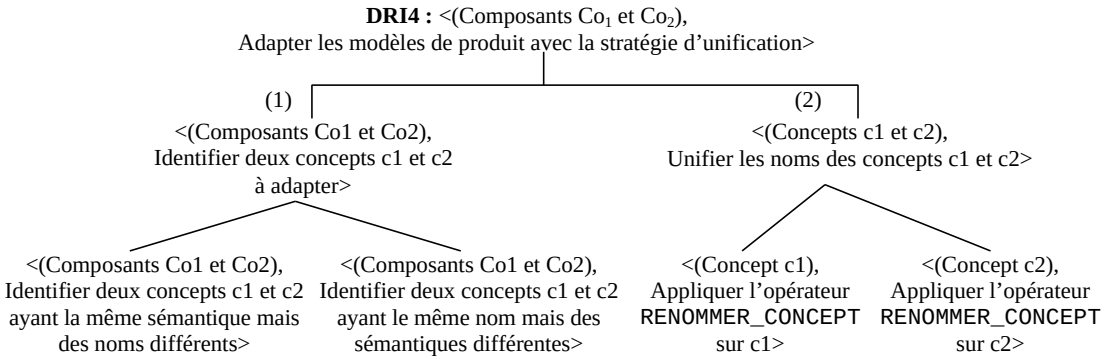


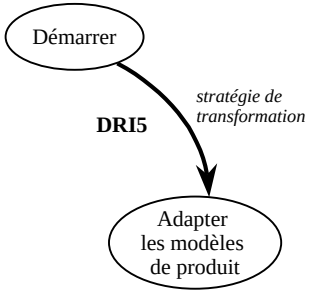
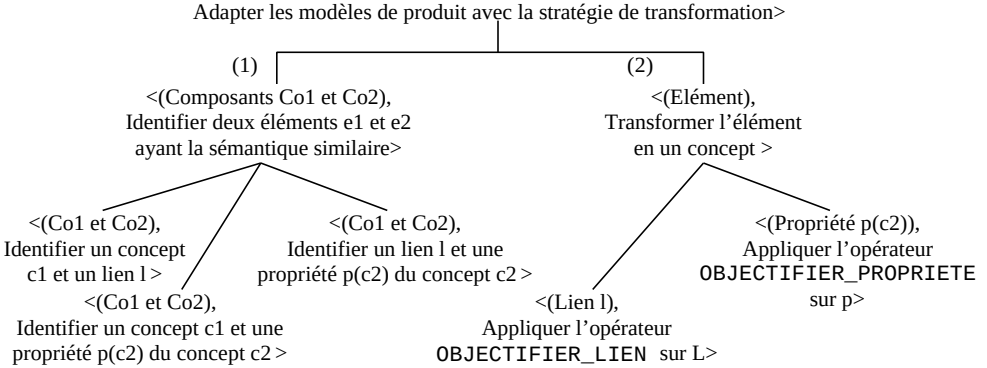
Critères de choix :
c1 : Il existe des éléments à adapter dans les modèles de processus des composants.
c2 : Il existe un couple d'intentions des cartes différentes ayant la même sémantique et le même nom.
c3 : Il existe des éléments à adapter dans les modèles de produit des composants.
c4 : Il existe un couple d'éléments des deux modèles de produits prêts à être intégrés.</td></tr>
<tr><td>2</td><td> 

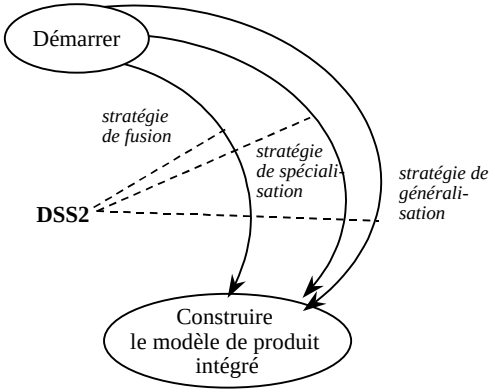
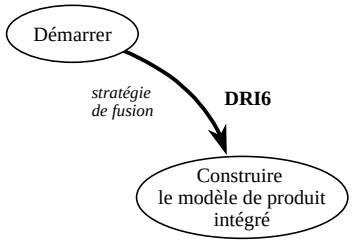
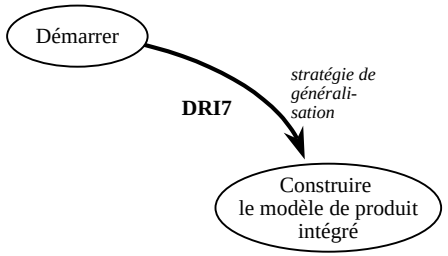


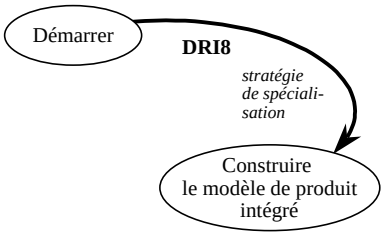
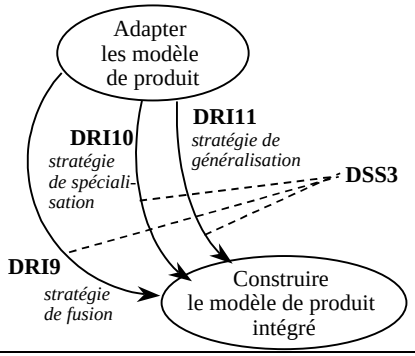
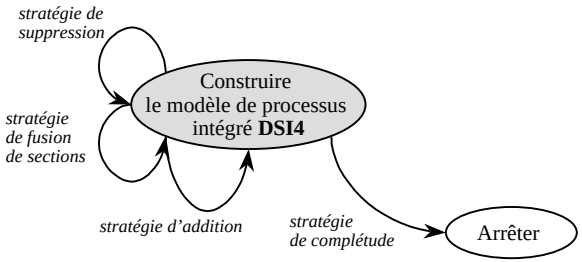
(1) Appliquer la mesure de similarité des cartes ASI. Toutes les mesures de similarité utilisées dans ce modèle de processus sont décrites dans le Chapitre 5 de ce mémoire.
(2) Appliquer l'opérateur RENOMMER_INTENTION sur une des deux intentions.</td></tr>
</table>

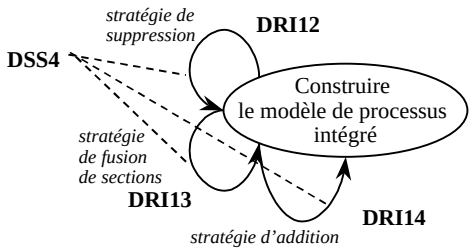
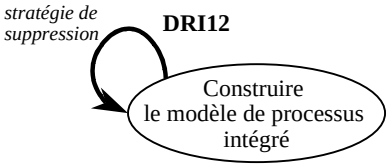
3		**DRI2 :** <(Composants Co1 et Co2), Construire le modèle de processus intégré avec la stratégie de fusion> (1) <(Composants Co1 et Co2), Identifier deux intentions I_{1i} et I_{2j} identiques> (2) <(Intentions I_{1i} et I_{2j}), Appliquer l'opérateur `FUSIONNER_INTENTION` sur I_{1i} et I_{2j}> (1) Appliquer la mesure de similarité des cartes ASI afin d'identifier deux intentions ayant les mêmes noms et des sémantiques similaires. (2) Appliquer l'opérateur `FUSIONNER_INTENTION` sur les deux intentions sélectionnées.
4		**DRI3 :** <(Composants Co1 et Co2), Construire un modèle de processus intégré avec la stratégie de fusion > La DRI3 est identique à la DRI2.
5		**DSS1 :** <(Composants Co1 et Co2), Progresser vers Adapter les modèles de produit > c1 : <(Composants Co1 et Co2), Sélectionner DRI4 :<(Co1 et Co2), Adapter les modèles de produit avec la stratégie d'unification> c2 : <(Composants Coo1 et Co2), Sélectionner DRI5 :<(Co1 et Co2), Adapter les modèles de produit avec la stratégie de transformation> Critères de choix : c1 : Il existe au moins deux concepts appartenant aux composants différents ayant des noms différents mais la même sémantique ou ayant le même nom mais des sémantiques différentes. c2 : Il existe au moins deux éléments représentant le même phénomène du monde réel mais exprimés par des moyens différents dans des composants différents.

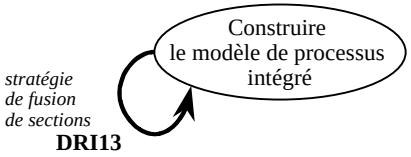
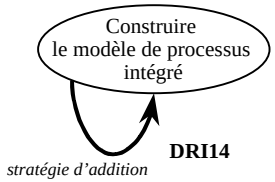
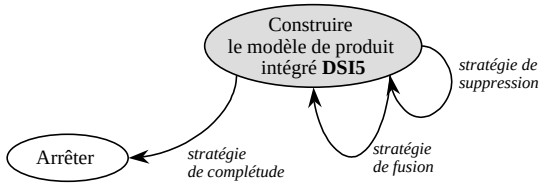
<table>
<tr>
<td>6</td>
<td> 
</td>
<td> 

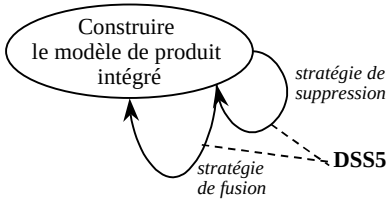
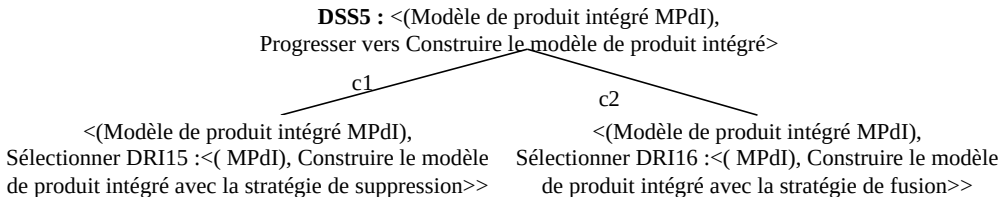
(1) Mesurer la similarité sémantique des concepts c1 et c2 avec la mesure AN. Mesurer la similarité structurelle des concepts c1 et c2 avec la mesure ASG.
(2) Appliquer l'opérateur RENOMMER_CONCEPT sur l'un des deux concepts si leurs noms sont différents mais leur sémantique et leur structure similaires ou si les noms sont identiques mais la sémantique différente.</td>
</tr>
<tr>
<td>7</td>
<td> 
</td>
<td> 
</td>
</tr>
</table>

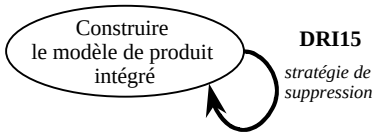
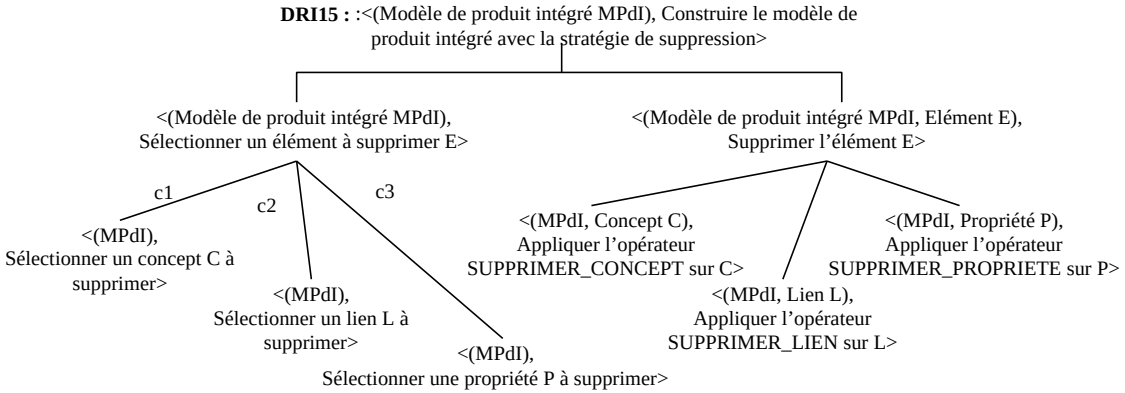
8		**DSS2 :** <(Composants Co1 et Co2), Progresser vers Construire le modèle de produit intégré> c1 : <(Composants Co1 et Co2), Sélectionner DRI2 :<(Composants Co1 et Co2), Construire le modèle de produit intégré avec la stratégie de fusion> c2 : <(Composants Co1 et oC2), Sélectionner DRI3 :<(Composants Co1 et Co2), Construire le modèle de produit intégré avec la stratégie de généralisation> c3 : <(Composants Co1 et Co2), Sélectionner DRI4 :<(Composants Co1 et Co2), Construire le modèle de produit intégré avec la stratégie de spécialisation> c1 : Il existe deux composants ayant le même nom, la même sémantique et des structures similaires. c2 : Il existe deux composants ayant la même sémantique mais des structures différentes. c3 : Il existe deux composants ayant la même sémantique mais la structure d'un composant est un sous-ensemble de la structure du deuxième.
9		**DRI6 :** <(Composants Co_1 et Co_2), Construire le modèle de produit intégré avec la stratégie de fusion> (1) <(Composants Co1 et Co2), Identifier deux concepts c1 et c2 ayant le même nom et les structures similaires> (2) <(Concepts c1 et c2), Appliquer l'opérateur `FUSIONNER_CONCEPT` sur c1 et c2> (1) Mesurer la similarité sémantique des concepts c1 et c2 avec la mesure AN. Mesurer la similarité structurelle des concepts c1 et c2 avec la mesure ASG. (2) Appliquer l'opérateur `FUSIONNER_CONCEPT` sur les deux concepts sélectionnés.
10		**DRI7 :** <(Composants Co_1 et Co_2), Construire le modèle de produit intégré avec la stratégie de généralisation> (1) <(Composants Co1 et Co2), Identifier deux concepts c1 et c2 ayant une sémantique similaire et une structure différente> (2) <(Concepts c1 et c2), Appliquer l'opérateur `GENERALISER` sur c1 et c2> (1) Mesurer la similarité sémantique des concepts c1 et c2 avec la mesure AN. Mesurer la similarité

		structurelle des concepts c1 et c2 avec la mesure ASG. (2) Appliquer l'opérateur `GENERALISER` sur les deux concepts sélectionnés.
11		**DRI8 :** <(Composants Co_1 et Co_2), Construire le modèle de produit intégré avec la stratégie de spécialisation> (1) <(Composants Co1 et Co2), Identifier deux concepts c1 et c2 ayant la sémantique similaire et la structure différente> (2) <(Concepts c1 et c2), Appliquer l'opérateur `SPECIALISER` sur c1 et c2> (1) Mesurer la similarité sémantique des concepts c1 et c2 avec la mesure AN. Mesurer la similarité structurelle des concepts c1 etc2 avec la mesure ASG. Le concept c2 doit avoir toutes les propriétés structurelles du concept c2. (2) Appliquer l'opérateur `SPECIALISER` sur les deux concepts sélectionnés.
12		La DSS3 est identique à la DSS2 (voir la situation 8 de ce tableau). La DRI9 est identique à la DRI6 (voir la situation 9 de ce tableau). La DRI10 est identique à la DRI7 (voir la situation 10 de ce tableau). La DRI11 est identique à la DRI8 (voir la situation 11 de ce tableau).
13		**DSI4 :** <(Modèle de processus intégré MpcI), Progresser de Construire le modèle de processus intégré> c1 : <(Modèle de processus intégré MpcI), Sélectionner DSS4 :<(MpcI) Progresser vers Construire le modèle de processus intégré>> c2 : <(Modèle de processus intégré MpcI), Sélectionner DRI17 :<(MpcI), Arrêter le processus d'assemblage avec la stratégie de complétude> c1 : Toutes les intentions similaires ont été fusionnées. Il est nécessaire d'affiner le modèle de processus intégré.

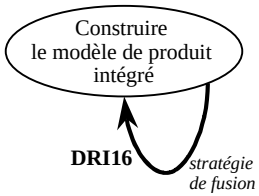
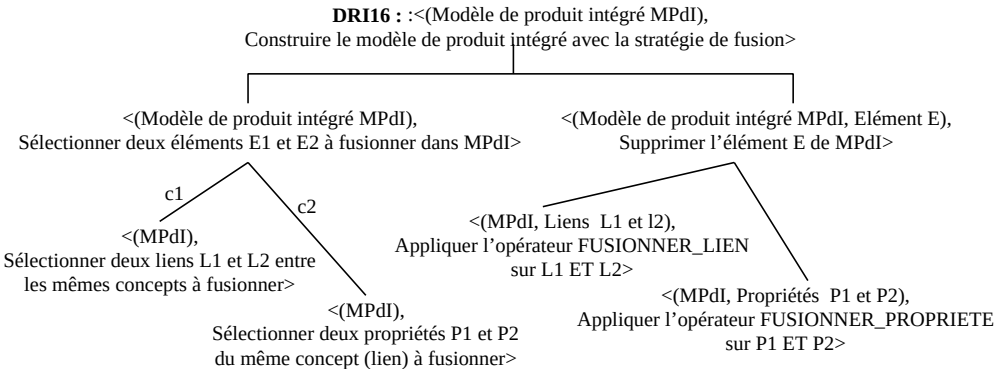
		c2 : L'ingénieur d'applications décide que la construction du modèle de processus intégré est terminée.
14		**DSS4 :** <(Modèle de processus intégré MpcI), Progresser vers Construire le modèle de processus intégré> c1 : <(Modèle de processus intégré MpcI), Sélectionner DRI12 :<(MpcI), Construire le modèle de processus intégré avec la stratégie de suppression>> c2 : <(Modèle de processus intégré MpcI), Sélectionner DRI13 :<(MpcI), Construire le modèle de processus intégré avec la stratégie de fusion>> c3 : <(Modèle de processus intégré MpcI), Sélectionner DRI14 :<(MpcI), Construire le modèle de processus intégré avec la stratégie d'addition>> Critères de choix : c1 = a1 OU a2 ; c2 = a3 ; c3 = a4 OU a5 Arguments : a1 : Il existe une section dans la carte intégrée qui n'est plus pertinente dans le processus intégré. a2 : La suppression préalable d'un concept (ou d'un lien entre les concepts) dans le modèle de produit intégré oblige de supprimer la (ou les sections) qui utilise(nt) ce concept (ou lien). a3 : Il existe deux sections similaires dans la carte de processus intégré. a4 : Il est nécessaire de faire la connexion entres deux intentions d'origine de cartes différentes dans la carte de processus intégré. a5 : Si le modèle de produit intégré permet la cohabitation de deux concepts ayant la même sémantique mais des structures différentes, il est nécessaire d'ajouter une section dans la carte intégrée permettant de transformer une instance d'un concept en une instance de l'autre concept.
15		**DRI12** : <(Modèle de processus intégré MPcI), Construire le modèle de processus intégré avec la stratégie de suppression> <(MPcI), Identifier une section <Ii, Ij, Sij> à supprimer> <(MPcI, <Ii, Ij, Sij>), Appliquer l'opérateur SUPPRIMER_SECTION sur <Ii, Ij, Sij>>

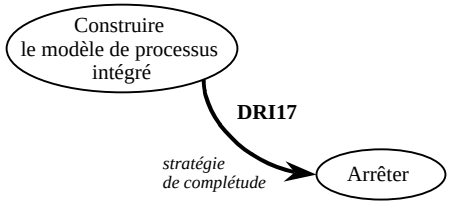
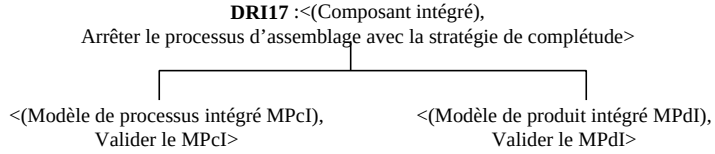
16		**DRI13** :<(Modèle de processus intégré MPcI), Construire le modèle de processus intégré avec la stratégie de fusion> <(MPcI), Identifier deux sections <Ii, Ij, Sij1> et <Ii, Ij, Sij2> entre Ii et Ij de MPcI>* <(MPcI, <Ii, Ij, Sij1>, <Ii, Ij, Sij2>), Mesurer la similarité des sections <Ii, Ij, Sij1> et <Ii, Ij, Sij2> avec SSS> c1 c2 <(MPcI, <Ii, Ij, Sij1>, <Ii, Ij, Sij2>), Garder les deux sections <Ii, Ij, Sij1> et <Ii, Ij, Sij2>> <(MPcI, <Ii, Ij, Sij1>, <Ii, Ij, Sij2>), Appliquer l'opérateur FUSIONNER_SECTION sur <Ii, Ij, Sij1> et <Ii, Ij, Sij2> > c1 : la valeur de SSS montre que les deux sections proposent des processus différents ; c2 : la valeur de SSS montre que les deux sections proposent des processus similaires ou identiques.
17		**DRI14** :<(Modèle de processus intégré MPcI), Construire le modèle de processus intégré avec la stratégie d'addition> <(MPcI), Identifier deux intentions Ii et Ij de MpcI à connecter>* <(MPcI, Ii, Ij,), Identifier une stratégie de connexion Sij entre Ii et Ij> <(MPcI, Ii, Ij, Sij), Appliquer l'opérateur AJOUTER_SECTION sur Ii, Ij et Sij1>
18		**DSI5 :** <(Modèle de produit intégré MPdI), Progresser de Construire le modèle de produit intégré> c1 c2 <(Modèle de produit intégré MPdI), Sélectionner DSS5 :<(MPdI), Progresser vers Construire le modèle de produit intégré>> <(Modèle de produit intégré MPdI), Sélectionner DRI18 :<(MPdI), Arrêter le processus d'intégration avec la stratégie de complétude>> c1 : Tous les éléments communs ont déjà été intégrés. L'ingénieur d'applications décide d'affiner le modèle de produit intégré. c2 : L'ingénieur d'applications décide que la construction du modèle de produit intégré est terminée.

<table>
<tr>
<td>19</td>
<td> 
</td>
<td> 

c1 : Il existe un concept, un lien ou une propriété qui n’a plus de sens dans le modèle de produit intégré.
c2 : Il existe deux liens ou deux propriétés similaires dans le modèle de produit intégré.</td>
</tr>
<tr>
<td>20</td>
<td> 
</td>
<td> 

c1: Après la suppression d’une section il existe un concept qui n’est plus utilisé par le modèle de processus.
c2 : Il existe un lien qui n’a plus de sens dans le modèle de produit intégré.
c3 : Après la fusion des concepts (ou des liens) il existe une propriété structurelle qui n’est plus utile.</td>
</tr>
</table>

<table>
<tr>
<td>21</td>
<td> 
</td>
<td> 

c1: Après la fusion des concepts il existe deux liens similaires entre les mêmes concepts.
c2 : Après la fusion des concepts (des liens) il existe deux propriétés similaires.</td>
</tr>
<tr>
<td>22</td>
<td> 
</td>
<td> 

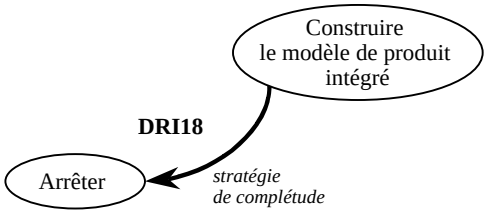
Valider le modèle de processus et le modèle de produit intégrés en appliquant les règles de cohérence et de complétude de modèle de processus.</td>
</tr>
<tr>
<td>23</td>
<td> 
</td>
<td>La DRI18 est identique à la DRI17.</td>
</tr>
</table>

Tableau 5 : Directives associées à la carte d'assemblage par intégration

Description

L'assemblage par intégration consiste à identifier des éléments communs à deux composants et à les fusionner ou les faire cohabiter dans le même modèle. Les cartes de processus de tels composants doivent avoir des intentions identiques ou similaires ; leurs modèles de produit doivent conceptualiser les mêmes objets du monde réel par des concepts identiques ou similaires.

Situation 1 : Comme le montre la carte de processus d'assemblage par intégration présentée à la Figure 127, nous proposons deux solutions pour démarrer l'intégration des composants donnés : (1) commencer l'assemblage des composants de méthode par l'intégration des modèles de processus et intégrer ensuite les modèles de produit en fonction du résultat obtenu, (2) commencer l'assemblage par l'intégration des modèles de produit puis intégrer les modèles de processus en fonction du résultat obtenu.

Situation 2 : Afin de pouvoir intégrer deux cartes, il faut souvent les adapter au préalable. L'intégration des cartes dans ce cas d'assemblage se fait par la fusion des intentions similaires, c'est-à-dire la fusion des intentions ayant non seulement des noms identiques mais aussi des sémantiques similaires. Si les noms de telles intentions sont différents, une des deux intentions doit être renommée afin de permettre leur fusion.

La DRI1 associée à la section <*Démarrer, Adapter les modèles de processus, Stratégie de fusion*> propose tout d'abord d'identifier un couple d'intentions de deux cartes initiales ayant des sémantiques similaires mais des noms différents (1.2) et de renommer une des intentions ensuite (2) en appliquant l'opérateur d'assemblage `RENOMMER_INTENTION`. Le coefficient d'affinité sémantique des intentions doit être égal à 1 pour ces deux intentions (voir ASI dans le Chapitre 5). La même directive prend également en compte le cas contraire, quand les intentions ont des noms identiques mais que leurs sémantiques sont différentes (1.1). Dans ce cas, une des intentions doit aussi être renommée afin d'interdire leur fusion. L'opérateur `RENOMMER_INTENTION` ainsi que d'autres opérateurs utilisés dans les processus d'assemblage sont présentés au Chapitre 5.

Situation 3 : Si les deux cartes initiales contiennent des intentions identiques, c'est-à-dire ayant les mêmes noms et des sémantiques similaires, ces stratégies peuvent être fusionnées dés le départ sans passer par l'étape d'adaptation.

La DRI2 associée à la section <*Démarrer, Construire le modèle de processus intégré, Stratégie de fusion*> propose tout d'abord d'identifier un couple d'intentions de deux cartes initiales ayant des sémantiques similaires à l'aide de la mesure d'affinité sémantique des intentions ASI (qui doit être égal à 1). Puis, l'ingénieur d'applications est invité à fusionner les deux intentions à l'aide de l'opérateur `FUSIONNER_INTENTION`.

Cette directive peut être appliquée autant de fois que l'on identifie des couples d'intentions similaires.

Situation 4 : La DRI3 associée à la section <*Adapter les modèles de processus, Construire le modèle de processus intégré, Stratégie de fusion*> est identique à la DRI2. Elle applique également l'opérateur `FUSIONNER_INTENTION.`

Situation 5 : La construction du modèle de produit intégré peut également être direct ou précédé par l'étape d'adaptation. La directive de progression de l'intention *Démarrer* vers l'intention *Adapter les modèles de produit* propose deux stratégies différentes pour adapter les éléments communs des modèles de produit : la stratégie d'unification et la stratégie de transformation.

La *stratégie d'unification* peut être utilisée pour résoudre le problème d'ambiguïté entre les concepts des différents modèles de produit car ceux-ci peuvent comporter des concepts ayant des noms différents mais de même sémantique ou au contraire, le même nom mais des sémantiques différentes. La DRI correspondante est présentée à la situation 6 du Tableau 5.

La *stratégie de transformation* traite le problème de modélisation quand les mêmes phénomènes du monde réel sont modélisés différemment dans les modèles de produit différents. La DRI correspondante est présentée à la situation 7 du Tableau 5.

Situation 6 : La DRI4 associée à la section <*Démarrer, Adapter les modèles de produit, Stratégie d'unification*> sert à l'unification de la terminologie des modèles de produit à intégrer. Afin de pouvoir fusionner des concepts, ce qui est le but de ce type d'assemblage, il est nécessaire d'avoir des couples de concepts ayant la même sémantique et le même nom. Les mesures de similarité AN (affinité des noms) et ASG (affinité structurelle globale) (Chapitre 5) doivent être calculées afin de pouvoir comparer les deux concepts sélectionnés. Si ces calculs montrent que les concepts ont la même sémantique mais des noms différents ou au contraire les mêmes noms mais des sémantiques différentes l'opérateur `RENOMMER_CONCEPT` doit être appliqué sur l'un des concepts soit pour unifier les noms soit pour les différencier.

Situation 7 : La DRI5 de la section <*Démarrer, Adapter les modèles de produit, Stratégie de transformation*> sert également à la préparation du terrain pour l'intégration des modèles de produit. Elle permet de faire des modifications dans l'un des modèles de produit afin d'obtenir des concepts similaires qui, à leur tour, pourront servir à l'intégration des deux modèles de produit par la voie de l'une des trois stratégies d'intégration : la fusion, la spécialisation et la généralisation qui sont présentées à la situation 12 du Tableau 5.

Cette directive peut être appliquée lorsqu'un phénomène du monde réel est représenté de manière différente dans les deux modèles de produit à intégrer. Par exemple, il peut être représenté par un concept dans un modèle de produit et par un lien entre deux concepts, ou bien, par une propriété structurelle d'un autre concept dans un autre modèle. La directive propose d'appliquer soit l'opérateur `OBJECTIFIER_LIEN` soit `OBJECTIFIER_PROPRIETE` respectivement et d'obtenir ainsi deux concepts ayant une sémantique similaire.

Situation 8 : La construction du modèle de produit intégré est également possible sans passer par l'étape d'adaptation des modèles de produits à condition qu'il existe dès le départ au moins un couple de concepts des modèles différents qui puissent être fusionnés, généralisés ou spécialisés. La directive de progression DSS2 propose trois stratégies différentes pour intégrer les éléments communs des modèles de produit :

La *stratégie de fusion* invite l'ingénieur d'applications à fusionner deux concepts de modèles différents ayant une sémantique et une structure similaires. Elle est présentée à la situation 9 du Tableau 5.

La *stratégie de généralisation* est basée sur l'introduction d'un nouveau concept qui représente la généralisation de deux concepts de modèles différents. Cette stratégie peut être appliquée si les deux concepts ont la même sémantique mais une structure différente. Elle est présentée à la situation 10 du Tableau 5.

Finalement, la *stratégie de spécialisation*, propose d'introduire un lien de spécialisation entre un concept d'un modèle de produit et un concept d'un autre modèle de produit si celui-ci a la même sémantique que le premier mais est structurellement plus spécifique. La DRI associée à cette section de la carte d'assemblage est représentée à la situation 11 du Tableau 5.

Situation 9 :. La DRI6 associée à la section <*Démarrer, Construire le modèle de produit intégré, Stratégie de fusion*> suggère d'identifier un couple de concepts similaires tant du point de vue sémantique que structurel et ayant le même nom. Les mesures de similarité AN (affinité des noms) et ASG (affinité structurelle globale) *(Chapitre 5)* doivent être calculées afin de pouvoir comparer les deux concepts sélectionnés. Si les résultats de ces calculs le permettent, les deux concepts peuvent être fusionnés à l'aide de l'opérateur `FUSIONNER_CONCEPT`.

Situation 10 : La DRI7 associée à la section <*Démarrer, Construire le modèle de produit intégré, Stratégie de généralisation*> est appliquée lorsque deux concepts ont la même sémantique mais que la mesure de ASG a montré que leurs structures sont trop différentes pour pouvoir les fusionner. La directive suggère de garder les deux concepts dans le nouveau modèle de produit en les généralisant en un nouveau concept de connexion à l'aide de l'opérateur `GENERALISER`. Les deux concepts doivent avoir des noms différents avant leur généralisation.

Situation 11 : La DRI8 de la section <*Démarrer, Construire le modèle de produit intégré, Stratégie de spécialisation*> est basée sur l'application de l'opérateur `SPECIALISER`. L'un des deux concepts faisant partie de la connexion entre deux modèles de produit doit avoir une structure plus riche que l'autre. Par conséquent, il peut être défini en tant que spécialisation du deuxième. Comme dans la situation 5, les deux concepts doivent avoir des noms différents.

Situation 12 : Une fois que toutes les adaptations des modèles de produit sont faites, l'ingénieur d'applications est invité par la DSS2 à progresser vers la construction du modèle de produit intégré. Le corps de la directive de progression DSS2 est identique à celui de la directive DSS3 car elle propose exactement les mêmes progressions, c'est-à-dire appliquer une des trois intégrations possibles : fusion, généralisation ou spécialisation des concepts similaires.

Il en est de même pour les DRI9, 10 et 11 qui correspondent aux DRI6, 7 et 8 respectivement.

Situation 13 : Une fois que l'ingénieur d'applications a fusionné toutes les intentions similaires, la carte lui offre le choix entre deux possibilités :

(1) itérer sur l'intégration des cartes pour affiner le résultat obtenu (situation 14) ou

(2) décider que l'intégration des modèles de processus est terminée et vérifier la complétude du résultat obtenu (situation 19 du Tableau 5).

Bien sûr, à tout moment l'ingénieur peut passer à l'affinement du modèle de produit intégré en fonction de l'état d'intégration des cartes.

Situation 14 : La carte de processus d'assemblage (Figure 127) propose trois stratégies pour affiner le modèle de processus intégré : la *stratégie de suppression*, la *stratégie de fusion* et la *stratégie d'addition*.

Certaines directives d'un composant peuvent être éliminées de la démarche intégrée pour des raisons différentes. Par exemple, on peut éliminer les directives qui ne sont pas assez détaillées ou sont mal définies. Il peut exister des directives dans la démarche du deuxième composant qui les remplacent. La *stratégie de suppression* peut être choisie pour résoudre ces problèmes.

Si l'ingénieur d'applications suspecte des sections parallèles similaires il peut choisir la *stratégie de fusion* afin de fusionner ces sections si nécessaire.

Il est parfois nécessaire de faire la connexion entres deux intentions d'origine de cartes différentes dans la carte de processus intégré. Si, par exemple, le modèle de produit intégré permet la cohabitation de deux concepts ayant la même sémantique mais des structures différentes, il est nécessaire d'ajouter une section dans la carte intégrée permettant de transformer une instance d'un concept en une instance de l'autre concept. Dans ce cas, l'ingénieur d'applications est invité à choisir la *stratégie d'addition*.

Situation 15 : La DRI12 associée à la section <*Construire le modèle de processus intégré, Construire le modèle de processus intégré, Stratégie de suppression*> suggère de supprimer les sections qui ne sont plus pertinentes après l'intégration des deux cartes. Une section peut devenir inutile quand l'intégration de la carte à laquelle elle appartient avec une autre carte apporte une autre section répondant aux mêmes besoins mais proposant un processus plus riche. La directive applique l'opérateur `SUPPRIMER_SECTION`. La suppression d'un concept dans le modèle de produit intégré peut également impliquer la suppression d'une ou plusieurs sections utilisant ce concept. L'opérateur `SUPPRIMER_SECTION` est appliqué pour résoudre ce problème.

Situation 16 : La DRI13 associée à la section <*Construire le modèle de processus intégré, Construire le modèle de processus intégré, Stratégie de fusion*> aide à identifier un couple de sections parallèles similaires telles que l'existence d'une des deux est superflue. Pour cela elle propose de comparer les deux sections à l'aide de la mesure de similarité sémantique des sections (SSS) et de fusionner ses sections à l'aide de l'opérateur `FUSIONNER_SECTION` si elles sont effectivement similaires, ou les garder toutes les deux dans le cas contraire.

Situation 17 : La DRI14 associée à la section <*Construire le modèle de processus intégré, Construire le modèle de processus intégré, Stratégie d'addition*> permet d'identifier un couple d'intentions dans la carte intégrée qui nécessitent d'être connectées par une stratégie de passage d'une intention vers une autre. Par exemple, si le modèle de produit intégré contient deux concepts ayant la même sémantique mais une structure différente, la carte de processus doit permettre de transformer le produit qui est

instance d'un concept en un produit qui soit instance de l'autre concept. La DRI14 propose d'ajouter une section de connexion entre deux intentions d'origine de cartes différentes en appliquant l'opérateur `AJOUTER_SECTION`.

Situation 18 : Une fois que tous les éléments communs des modèles de produit sont intégrés, l'ingénieur d'applications a deux possibilités :

1. affiner le modèle de produit obtenu (situation 19) ou

2. arrêter le processus de construction du composant intégré en validant sa complétude (situation 23 du Tableau 5).

Situation 19 : L'affinement du modèle de produit intégré est souvent nécessaire afin d'obtenir un modèle de produit complet et cohérent.

Il existe parfois des concepts, des liens ou des propriétés qui pour différentes raisons ne doivent plus exister dans le modèle de produit intégré. Par exemple, la suppression d'une section dans le modèle de processus intégré peut impliquer la suppression d'un concept ou d'un lien si celui-ci n'était utilisé que par cette section. La DSS5 propose de sélectionner la DRI15 avec l'objectif de le supprimer *(situation 20).*

La fusion des concepts peut amener à une situation où certains liens ou propriétés existent en double dans le modèle de produit intégré. La DSS5 propose dans ce cas de sélectionner la DRI16 afin de pouvoir les fusionner (situation 21).

Situation 20 : La DRI15 aide l'ingénieur d'applications à identifier un élément (un concept, un lien ou une propriété) qui n'a plus aucune utilité dans le modèle de produit intégré et de l'éliminer en appliquant l'opérateur `SUPPRIMER_CONCEPT`, `SUPPRIMER_LIEN` ou `SUPPRIMER_PROPRIETE` selon le cas.

Situation 21 : La DRI16 permet d'identifier des éléments (des liens ou des propriétés) qui existent en double dans le modèle de produit intégré grâce aux mesures de similarité. Elle propose de fusionner ces éléments en appliquant l'opérateur `FUSIONNER_LIEN` ou `FUSIONNER_PROPRIETE` suivant le cas.

Situations 22 - 23 : Les DRI17 et 18 sont destinées à la validation de l'intégration obtenue. Elles vérifient la cohérence et la complétude du modèle de produit et du modèle de processus intégrés.

3.4.3 Exemple d'assemblage des composants par intégration

Nous allons illustrer l'application du composant d'assemblage CA1 avec un exemple d'intégration d'un composant issu de la méthode OOSE [Jacobson 92], [Jacobson 95] qui permet de découvrir des besoins fonctionnels d'un système d'information sous forme d'un modèle des cas d'utilisation et d'un composant issu de la méthode CREWS-*L'Ecritoire* [Rolland 98b], [Ben Achour 98], [Tawbi 99b] qui permet également de découvrir les besoins fonctionnels d'un système d'information sous forme des

couples <But, Scénario>. Nous les désignons respectivement *composant* OOSE et *composant* CREWS-*L'Ecritoire.* Avant de procéder à l'intégration de ces composants de méthode, nous présentons d'abord leurs contenus, c'est-à-dire leurs modèles de produit et leurs modèles de processus respectifs en les comparant en même temps.

3.4.3.1 Modèles de produit de CREWS-*L'Ecritoire et de* OOSE

3.4.3.1.1 Modèle de produit du composant OOSE

Le modèle des cas d'utilisation de la méthode OOSE est destiné à la description du comportement d'un système du point de vue de ses utilisateurs. Le modèle est un ensemble de cas d'utilisation où chaque cas d'utilisation décrit un service proposé par le système à ses utilisateurs. Le modèle de produit est illustré à la Figure 128.

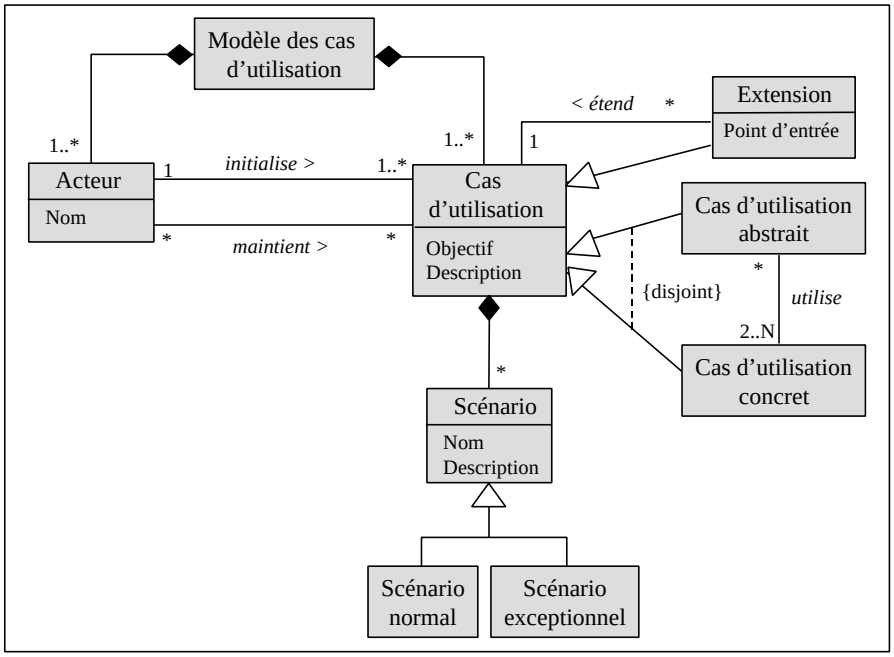


Figure 128 : Le modèle de produit de la méthode OOSE

Comme le montre la Figure 128, le concept principal du modèle des cas d'utilisation de OOSE est *un cas d'utilisation*. Un cas d'utilisation est composé d'un ensemble de *scénarios*. Il a toujours un *scénario normal* et plusieurs *scénarios d'exception*. Contrairement au *scénario* du composant CREWS-*L'Ecritoire* (voir sous-section suivante), le scénario n'est pas décomposé en sous-éléments ; il s'agit d'une description informelle.

Le composant permet la réutilisation des descriptions communes à plusieurs cas d'utilisations par le biais des cas d'utilisation abstraits. Un cas d'utilisation peut être *concret* ou *abstrait*. Les cas abstraits sont des extractions des descriptions communes à plusieurs cas concrets. Ils ne peuvent pas être instanciés en tant que tels car ils ne décrivent pas des scénarios complets mais seulement des fragments des scénarios qui sont partagés par plusieurs cas. Ils permettent de réutiliser ces descriptions communes dans celles de nouveaux cas concrets. Le composant permet aussi d'étendre les cas

d'utilisation par des extensions optionnelles définies par des *cas d'extension* qui sont aussi considérés comme des cas d'utilisation.

Chaque cas d'utilisation a un *acteur* qui l'initialise en interagissant avec le système. Il peut aussi avoir un ou plusieurs acteurs qui interviennent au cours de l'exécution d'un cas d'utilisation.

Finalement, le *modèle des cas d'utilisation* est défini comme une collection des cas d'utilisation initiés par des acteurs.

3.4.3.1.2 Modèle de produit du composant CREWS-*L'Ecritoire*

Le composant de méthode CREWS-*L'Ecritoire* est destiné à la découverte des besoins à partir de scénarios textuels. Le concept principal de l'approche est appelé un *Fragment de Besoin* (FB). Un Fragment de Besoin est un couple <*But*, *Scénario*>, où le *but* est défini comme "quelque chose que l'utilisateur du système espère obtenir dans le futur" et le *scénario* est défini comme "le comportement possible du système limité à un ensemble d'interactions significatives entre plusieurs agents pour atteindre le but" [Rolland 98b].

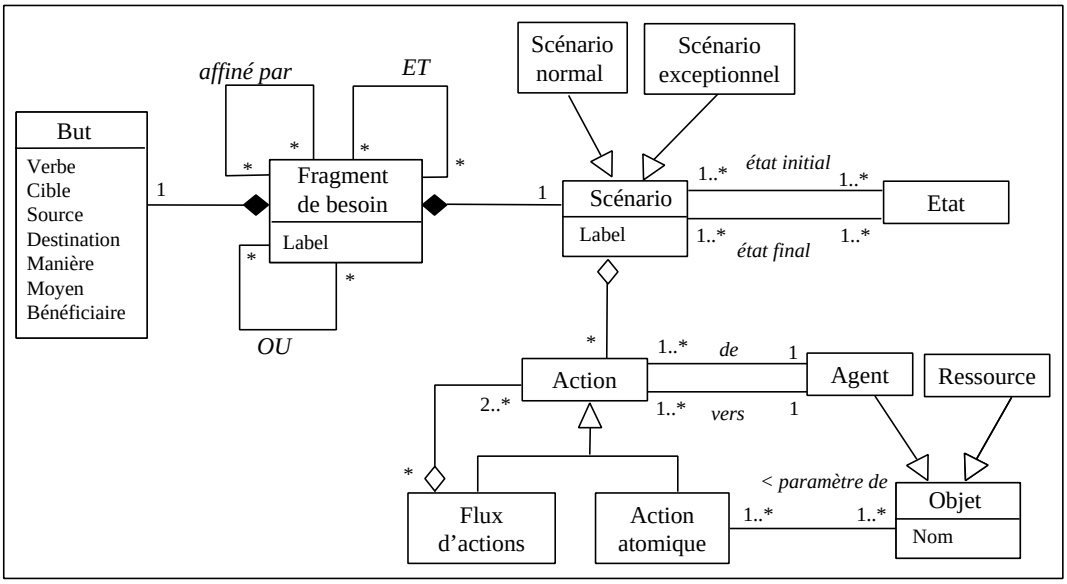


Figure 129 : Modèle de produit de l'approche CREWS-*L'Ecritoire*

Comme le montre la Figure 129 illustrant le modèle de produit de CREWS-*L'Ecritoire*, le but est composé d'un verbe et d'un ou plusieurs paramètres tels que cible, source, manière. Chaque paramètre joue un rôle particulier à l'égard du verbe.

Contrairement au composant OOSE, le scénario est structuré. Il est composé d'une ou plusieurs *actions*. Une combinaison des actions dans un scénario décrit un chemin unique menant des états initiaux aux états finaux des agents impliqués dans le scénario. Par conséquent, le comportement d'un système d'agents est décrit par un ensemble de scénarios.

Un scénario est caractérisé par un *état initial* et un *état final*. Un état initial attaché au scénario définit une pré-condition à satisfaire pour que l'exécution du scénario soit déclenchée. Par exemple, dans un système proposant des services de recyclage de différents articles tels que des bouteilles, des journaux etc., le scénario associé au but *"Déposer les bouteilles dans une machine de recyclage à carte dans le*

cas normal" ne peut être déclenché que si les états initiaux *"L'utilisateur a une carte"* et *"La machine de recyclage est prête"* sont vrais. Un état final définit l'état atteint à la fin du scénario. Par exemple, le scénario précédent mène aux états *"Le client a la carte", "La machine de recyclage est prête"* et *"Le client a un reçu"*.

Comme le composant OOSE, CREWS-*L'Ecritoire* propose deux types de scénario : le *scénario normal* et le *scénario exceptionnel*. Le scénario normal permet d'atteindre le but associé, tandis que le scénario exceptionnel décrit le cas où le but n'est pas atteint. Le scénario associé au but *"Déposer les bouteilles dans une machine de recyclage à carte dans le cas exceptionnel quand la carte n'est pas valide"* est un exemple de scénario exceptionnel, car le but *"Déposer les bouteilles dans une machine de recyclage à carte"* n'est pas atteint.

Le scénario peut être décrit en utilisant deux types *d'action* : *l'action atomique* et le *flux d'actions*. Une action atomique est une interaction entre deux agents qui affecte certains objets. Chaque agent et chaque objet de type ressource peuvent participer à plusieurs actions atomiques. La phrase *"Le client insère la bouteille dans la machine de recyclage"* est un exemple d'action atomique. Cette action implique deux agents *"Le client"* et *"La machine de recyclage"*. Le paramètre *"la bouteille"* est un objet ressource.

Un flux d'actions est composé de plusieurs actions. La phrase *"Si le client demande le reçu, la MR imprime le reçu"* est un exemple de flux d'actions comprenant deux actions atomiques. Le flux d'actions a l'une des sémantiques suivantes : séquence, alternative, répétition ou concurrence.

Les FBs sont organisés de manière hiérarchique à travers des relations de *composition*, *d'alternative* et *d'affinement*. Les relations de composition et d'alternative permettent respectivement de construire une structure *ET* et *OU* entre les FBs. La structure ET relie les FBs qui sont complémentaires les uns des autres pour définir la fonctionnalité complète du système (le modèle de cas d'utilisation dans OOSE). La structure OU relie les FBs alternatifs et représente ainsi les chemins alternatifs pour satisfaire le même but (le cas d'utilisation dans OOSE). La relation d'affinement organise les FBs en une hiérarchie des fragments de différents niveaux d'abstraction. Chaque interaction dans le scénario d'un certain niveau peut être vue comme un but à atteindre au niveau suivant.

Par conséquent, trois niveaux d'abstraction sont identifiés dans CREWS-*L'Ecritoire* : contextuel, fonctionnel et interne.

- Le *niveau contextuel* : le FB de niveau contextuel est composé d'un *but de conception* et d'un *scénario de service*. Un but de conception correspond à un choix de solution. Un scénario de service décrit des dépendances entre agents. Ces dépendances sont appelées *services*.

- Le *niveau fonctionnel*: un *but de service* et un *scénario fonctionnel* composent le FB *fonctionnel*. Un but de service définit un service que le système doit rendre à ses utilisateurs (une fonctionnalité du système). Un scénario *fonctionnel* décrit les interactions entre les agents pour satisfaire le but de service.

- Le *niveau interne* : un *but de système* et un *scénario interne* composent le FB interne. Un but de système exprime une manière possible pour exécuter une action identifiée dans le scénario

d'interaction. Un scénario interne décrit le flux d'interactions entre les objets du système pour satisfaire le but de ce système.

Puisque le modèle des cas d'utilisation de OOSE ne concerne que le niveau *fonctionnel* nous avons deux possibilités : limiter CREWS-*L'Ecritoire* au niveau fonctionnel ou bien l'utiliser en entier et enrichir ainsi le modèle des cas d'utilisation par le niveau contextuel et le niveau interne.

La représentation des deux modèles de produits montre que la structure des différents éléments est différente même si leur sémantique est similaire.

3.4.3.2 Modèles de processus de CREWS-*L'Ecritoire et de* OOSE

3.4.3.2.1 Modèle de processus du composant OOSE

La Figure 130 présente la carte de processus du composant OOSE. Cette carte représente le processus de construction du modèle des cas d'utilisation.

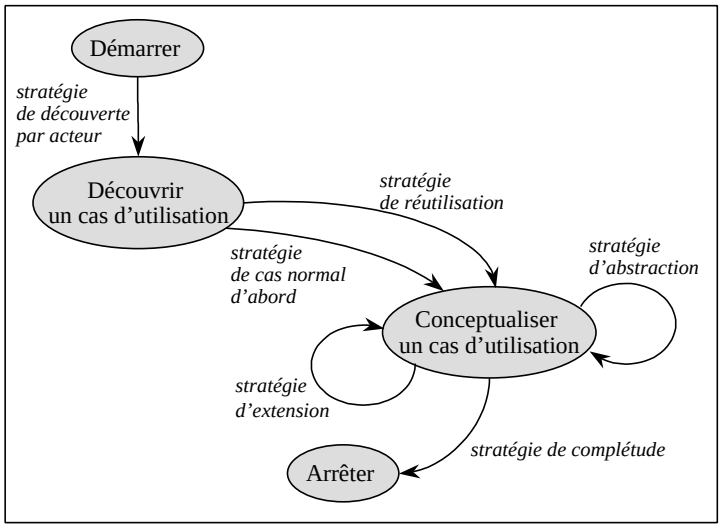


Figure 130 : Le modèle de processus du composant OOSE

Le petit nombre de stratégies proposées dans la carte du composant reflète la nature séquentielle du processus proposé par le composant. Par exemple, il ne fournit qu'une possibilité pour démarrer le processus de développement : découvrir des cas d'utilisation avec une stratégie *par acteur*. Le composant suggère d'identifier d'abord les acteurs du système comme le moyen d'identification des cas d'utilisation. Ensuite, deux sections sont proposées à l'ingénieur d'applications : <*Découvrir un cas d'utilisation, Conceptualiser un cas d'utilisation, Stratégie cas normal d'abord*> et <*Découvrir un cas d'utilisation, Conceptualiser un cas d'utilisation, Stratégie d'abstraction*>. Ces sections reflètent deux possibilités de conceptualisation des cas d'utilisation. La première est de conceptualiser chaque cas d'utilisation découvert précédemment en écrivant d'abord le scénario normal et ensuite tous les scénarios exceptionnels. La seconde est de conceptualiser un cas d'utilisation en réutilisant les descriptions de cas d'utilisation abstraits à condition qu'on en ait déjà construit précédemment.

Une fois l'intention *"Conceptualiser un cas d'utilisation"* d'utilisation réalisée, l'ingénieur d'applications peut conceptualiser d'autres cas d'utilisation en choisissant une section parmi les trois sections proposées : <*Conceptualiser un cas d'utilisation, Conceptualiser un cas d'utilisation, Stratégie d'abstraction*>, <*Conceptualiser un cas d'utilisation, Conceptualiser un cas d'utilisation, Stratégie d'extension*> ou <*Conceptualiser un cas d'utilisation, Conceptualiser un cas d'utilisation, Stratégie de complétude*>. La section <*Conceptualiser un cas d'utilisation, Conceptualiser un cas d'utilisation, Stratégie d'abstraction*> permet d'extraire un cas d'utilisation abstrait à partir d'un ensemble de cas d'utilisation concrets. La section <*Conceptualiser un cas d'utilisation, Conceptualiser un cas d'utilisation, Stratégie d'extension*> permet d'étendre un cas d'utilisation par des extensions. Et finalement la section <*Conceptualiser un cas d'utilisation, Conceptualiser un cas d'utilisation, Stratégie de complétude*> permet d'arrêter le processus de développement si le modèle des cas d'utilisation obtenu est complet.

3.4.3.2.2 Modèle de processus du composant CREWS-*L'Ecritoire*

La carte de processus du composant CREWS-*L'Ecritoire* décrite à la Figure 131 représente les processus qui permettent de conceptualiser un ensemble de scénarios décrivant les besoins fonctionnels d'un système. L'ensemble complet des scénarios obtenus en appliquant la méthode couvre l'ensemble des cas d'utilisations obtenus en utilisant le modèle des cas d'utilisation de la méthode OOSE.

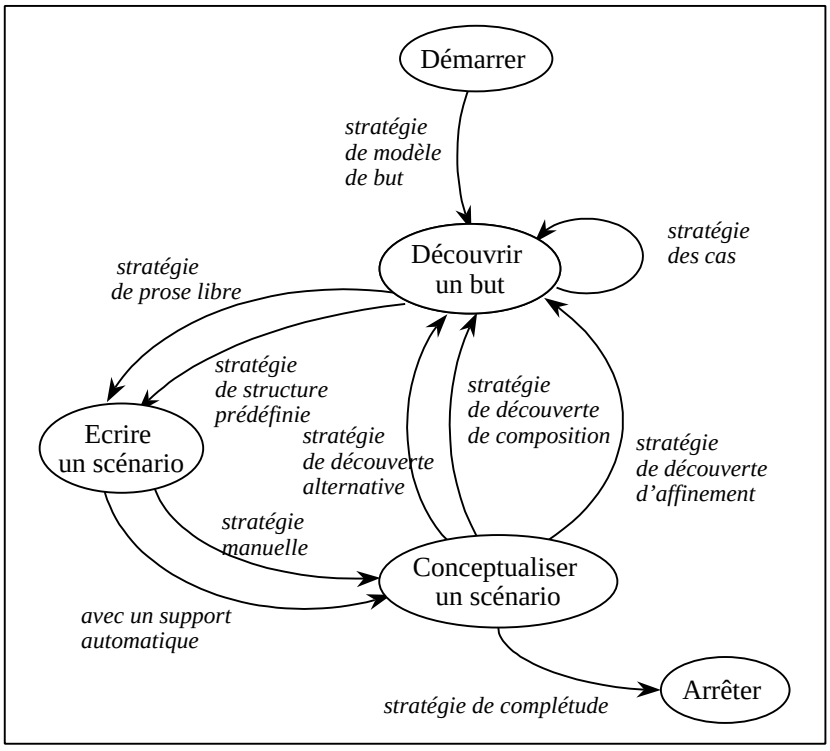


Figure 131 : Le modèle de processus de l'approche CREWS-*L'Ecritoire*

La découverte d'un but et l'écriture d'un scénario correspondant sont deux activités complémentaires dans le processus de découverte des besoins dans CREWS-*L'Ecritoire*. La séquence <*Découvrir un but, Ecrire un scénario*> est répétée pour construire une hiérarchie des fragments de besoin.

Comme le montre la Figure 131, la carte du composant CREWS-*L'Ecritoire* propose plusieurs stratégies pour réaliser les différentes intentions du processus. En conséquence, le processus proposé est plus riche et il est aussi plus flexible que celui de OOSE. Par exemple, la découverte d'un but peut être suivie par la découverte d'autres buts représentant des alternatives de conception de ce dernier ou par l'écriture du scénario correspondant. La *"stratégie des cas"* est proposée pour découvrir des buts alternatifs à celui de départ afin d'affiner la solution de conception du système. Cette stratégie propose de définir des valeurs alternatives à tous les paramètres du but, de construire des nouveaux buts en assemblant ces nouvelles valeurs et de sélectionner le but représentant la meilleure solution. L'écriture d'un scénario est prise en compte par deux stratégies appelées *"la stratégie de structure prédéfinie"* et *"la stratégie de prose libre"*. La première propose d'écrire un scénario en suivant une structure prédéfinie tandis que suivant la deuxième l'ingénieur d'applications écrit un scénario en prose libre. Dans ce cas les directives de style et de contenu sont proposées pour guider l'ingénieur d'applications dans l'écriture des scénarios.

L'écriture d'un scénario peut être suivie par sa conceptualisation. Pour cela, la carte propose deux stratégies: la *stratégie manuelle* et la *stratégie avec un support automatisé*, c'est-à-dire l'outil ***Ecritoire*** [Tawbi 99a], [Souveyet 98].

La conceptualisation du scénario peut être suivie par la découverte des nouveaux buts (progression vers l'intention *"Découvrir un but")* ou l'arrêt du processus de développement (progression vers l'intention *"Arrêter"*). Trois stratégies sont proposées à l'ingénieur d'applications pour découvrir des nouveaux buts : *"la stratégie de découverte alternative"*, *"la stratégie de découverte de composition"* et *"la stratégie de découverte d'affinement"*. La découverte des nouveaux buts en suivant *"la stratégie de découverte alternative"* permet d'identifier tous les buts alternatifs à un but donné. L'ensemble des scénarios correspondants contient un scénario de cas normal et tous les scénarios alternatifs et exceptionnels et par conséquent il correspond à un cas d'utilisation. La découverte des nouveaux buts en suivant *"la stratégie de découverte de composition"* permet d'identifier tous les buts complémentaires à un but donné et, en même temps, elle aide à identifier la famille des cas d'utilisation pour un système donné. Enfin, la découverte des nouveaux buts en suivant *"la stratégie de découverte d'affinement"* permet de descendre d'un niveau d'abstraction et d'affiner un but par un ensemble des buts de niveau d'abstraction plus bas.

Quand finalement, l'ingénieur d'applications décide d'arrêter le processus d'application, il choisit *"la stratégie de complétude"* qui aide à vérifier la cohérence et la complétude du modèle obtenu.

3.4.3.3 Justification de la fusion des composants de méthode OOSE *et* CREWS-*L'Ecritoire*

L'intégration de ces deux composants est justifiée par la possibilité de cumuler les avantages des deux composants tout en supprimant leurs inconvénients :

- Dans le composant CREWS-*L'Ecritoire* l'écriture des scénarios est soutenue par des directives de style et de contenu et des outils linguistiques tandis que le composant OOSE ne fournit aucune directive pour aider l'ingénieur d'applications à l'écriture des scénarios. Des études empiriques

[Tawbi 99a], [Tawbi 00] ont montré que les scénarios obtenus en appliquant les directives de CREWS-*L'Ecritoire* sont plus clairs et plus structurés qu'en l'absence de directives. Il y a donc avantage à les conserver.

- Le composant CREWS-*L'Ecritoire* guide la découverte systématique des variantes d'un cas d'utilisation contrairement au composant OOSE qui ne dit pas comment trouver tous les cas exceptionnels.

- Le composant CREWS-*L'Ecritoire* aide à atteindre la complétude de la famille des cas d'utilisation tandis que le composant OOSE ne dit pas comment vérifier si on a découvert tous les cas d'utilisation possibles.

- De plus, le composant CREWS-*L'Ecritoire* permet de descendre du niveau fonctionnel au niveau interne du système et de décrire les interactions entre les objets du système.

- Le composant OOSE propose un moyen de réutiliser les descriptions communes à plusieurs cas d'utilisation au lieu de les réécrire chaque fois.

- Enfin, le composant OOSE propose un moyen d'étendre le modèle des cas d'utilisation déjà complets en ajoutant de nouvelles fonctionnalités ou des options omises dans la première version du modèle.

Nous proposons d'assembler les deux composants de méthode pour obtenir un nouveau composant plus riche que ses deux ancêtres.

3.4.3.4 Démarche utilisée pour la fusion des composants OOSE *et* CREWS-*L'Ecritoire*

La carte de processus d'assemblage des composants par intégration présentée à la Figure 127 propose deux possibilités pour démarrer l'intégration des composants: commencer par l'intégration des modèles de processus des composants ou commencer par l'intégration de leurs modèles de produit.

Puisque c'est l'enrichissement du modèle de processus qui nous intéresse, nous commençons l'assemblage des composants par l'intégration de leurs cartes de processus en identifiant des intentions similaires susceptibles d'être fusionnées.

L'intention *"Découvrir un but"* dans la carte de CREWS-*L'Ecritoire* et l'intention *"Découvrir un cas d'utilisation"* dans la carte de OOSE ont des noms différents mais leurs sémantiques sont similaires. Les deux intentions se réfèrent à des fonctionnalités que le système doit fournir à ses utilisateurs. Une fonctionnalité du système est exprimée par la notion du *"but"* dans la méthode CREWS-*L'Ecritoire* tandis que la méthode OOSE utilise le terme de *"cas d'utilisation"*. Les deux intentions ont alors la même sémantique mais leurs noms sont différents. Afin de pouvoir les fusionner il faut passer par l'étape d'adaptation des cartes. On choisit alors la section <*Démarrer, Adapter les modèles de processus, Stratégie d'unification*> de la carte d'assemblage (Figure 127).

Puisque, le fait de découvrir un cas d'utilisation signifie l'identification d'un objectif que le système d'information doit satisfaire et dont la réalisation doit être décrite par un cas d'utilisation, nous décidons de garder le nom de l'intention *"Découvrir un but"* et de renommer l'intention *"Découvrir un cas d'utilisation"* en appliquant l'opérateur `RENOMMER_INTENTION`. La signature da la DSI associée à l'intention renommée doit aussi être renommée. Les détails d'application de cet opérateur sont répertoriés au Tableau 6.

Une fois les noms des intentions unifiés, celles-ci peuvent être fusionnées en sélectionnant la section *<Adapter les modèles de processus, Construire un modèle de processus intégré, Stratégie de fusion>* dans la carte d'assemblage (Figure 127). On applique l'opérateur `FUSIONNER_INTENTION` sur les deux intentions *"Découvrir un but"*. La Figure 132 montre le résultat d'application de cet opérateur. Toutes les sections des deux cartes desquelles l'intention *"Découvrir un but"* faisait partie comme intention cible ou intention source sont préservées dans la nouvelle carte.

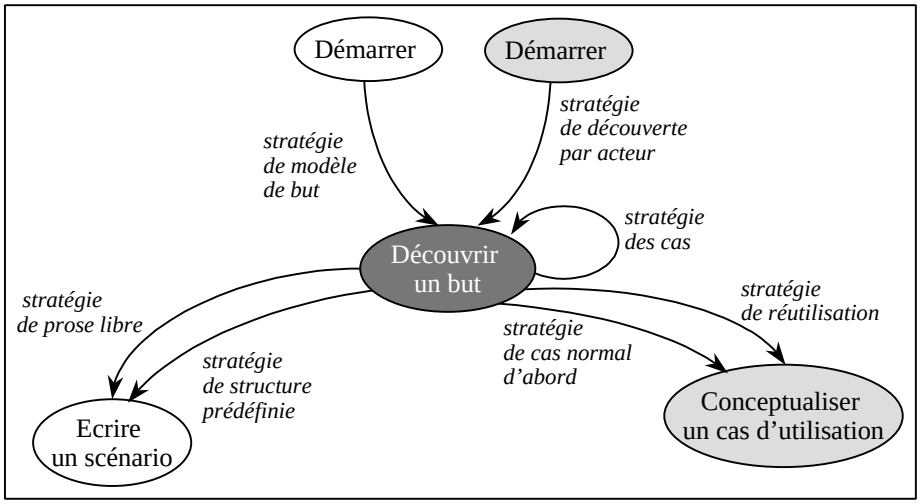


Figure 132 : Application de l'opérateur FUSIONNER_INTENTION sur l'intention "Découvrir un but"

L'opérateur `FUSIONNER_INTENTION` construit une nouvelle DSI associée à l'intention fusionnée *"Découvrir un but"* à partir des DSI des intentions initiales. Comme le montre la Figure 133, la nouvelle DSI propose la possibilité de progresser vers l'écriture d'un scénario suivant les stratégies proposées par le composant CREWS-*L'Ecritoire* et la possibilité de progresser vers la conceptualisation d'un cas d'utilisation suivant les stratégies proposées par le composant OOSE. Le Tableau 6 récapitule tous les opérateurs appliqués lors de l'intégration des deux composants.

DSI : <(But), Progresser à partir de Découvrir un but>	
<(But), Sélectionner DSS : <(But), Progresser vers Ecrire un scénario>>	<(But), Sélectionner DSS : <(But), Progresser vers Conceptualiser un cas d'utilisation>>

Figure 133 : La DSI associée à l'intention fusionnée "Découvrir un but"

De la même manière, nous appliquons l'opérateur `FUSIONNER_INTENTION` sur les intentions *"Démarrer"* et "*Arrêter"* des deux cartes. Ces deux intentions ne nécessitent pas d'être renommées avant leur fusion. C'est la section <*Démarrer, Construire un modèle de processus intégré, Stratégie de fusion*> de la Figure 127 qui est sélectionnée dans ces deux cas (Situations 3 et 4 au Tableau 6).

La fusion des intentions *"Démarrer"* construit une nouvelle DSI et une nouvelle DSS car dans les deux cartes on ne peut progresser que vers l'intention *"Découvrir un but"* à partir de l'intention "*Démarrer"*, seules les stratégies pour le faire sont différentes. Il nous faut alors construire une nouvelle DSS qui aiderait à la sélection d'une stratégie parmi les deux stratégies de départ (la "*stratégie de découverte par acteur"* de OOSE et la "*stratégie de modèle de but"* de CREWS-*L'Ecritoire*). La construction d'une telle directive consiste à définir les critères de choix argumentant la sélection de chaque stratégie. Par exemple, l'arguments en faveur de la "*stratégie de modèle de but"* serrait la possibilité de commencer la découverte des besoins du système à partir d'un objectif de conception très générale tandis que la sélection de la "*stratégie de découverte par acteur"* se justifierait par le fait que le système serait vu du point de vue d'utilisateur et que l'identification de ces futurs utilisateurs en premier permettrait l'identification des services que chacun d'eux attend du futur système. Les nouvelles DSI et DSS sont présentées à la Figure 134.

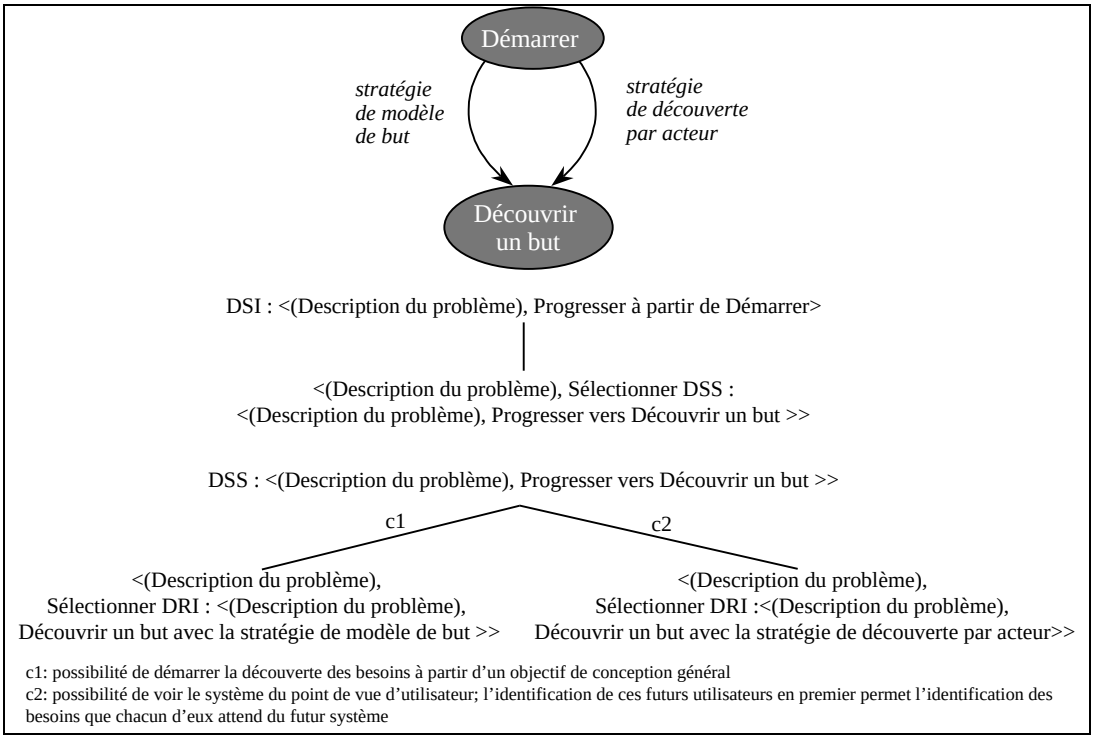


Figure 134 : Le résultat d'application de l'opérateur FUSIONNER_INTENTION sur les intentions "Démarrer"

Une fois que l'on a fusionné toutes les intentions possibles, on peut continuer l'intégration des cartes de processus en choisissant une des trois stratégies qui bouclent sur la construction du modèle de processus intégré avec l'objectif de l'affiner ou passer à l'intégration des éléments de produit avec un flux de stratégies à partir de l'intention *"Démarrer"* vers l'intention *"Adapter les modèles de produit"* ou directement vers l'intention *"Construire le modèle de produit intégré"* (Figure 127).

Supposons que l'on ait décidé de passer à l'intégration des modèles de produits. La fusion des deux intentions implique l'intégration des concepts correspondants dans le modèle de produit intégré. Elle peut être effectuée à l'aide d'un opérateur de fusion, de spécialisation ou de généralisation. De plus, la fusion des intentions *"Découvrir un but"* et *"Découvrir un cas d'utilisation"* demande de faire le lien entre les concepts correspondants dans les modèles de produit.

Le concept *"Cas d'utilisation"* n'existe que dans le composant OOSE. Cependant, l'ensemble de scénarios reliés à travers la relation *"OU"* dans CREWS-*L'Ecritoire* est équivalent à un cas d'utilisation. On choisit la section *<Démarrer, Adapter les modèles de produit, Stratégie de transformation>* (Figure 127) et on applique l'opérateur `OBJECTIFIER_LIEN` qui permet de transformer la relation *"OU"* entre un ensemble de *"Fragments de besoin"* en un nouveau concept que nous appelons un *"Cas d'utilisation"* (Figure 135). Maintenant nous pouvons fusionner les deux concepts *"Cas d'utilisation"* en un nouveau concept ayant le même nom. La DRI9 associée à la section *<Adapter les modèles de produit, Construire un modèle de produit intégré, Stratégie de fusion>* (Tableau 5) applique l'opérateur `FUSIONNER_CONCEPT` qui préserve tous les liens et toutes les propriétés des deux concepts initiaux.

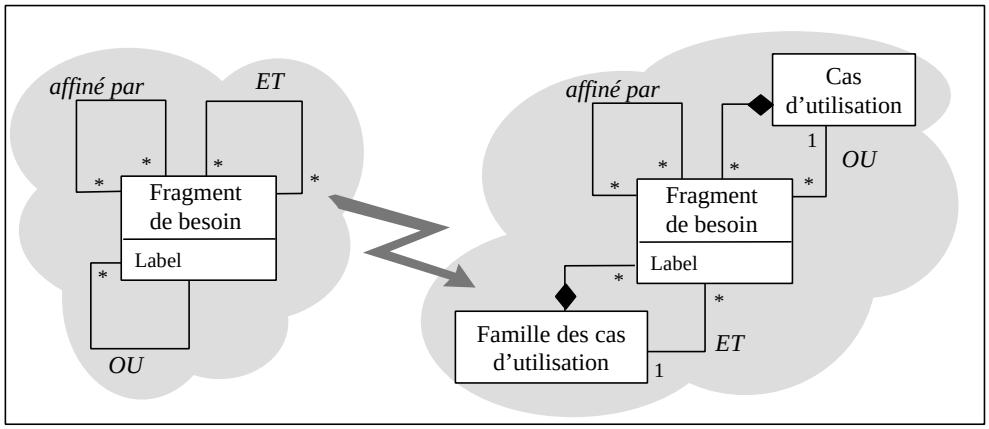


Figure 135 : Résultat d'application d'opérateur `OBJECTIFIER_LIEN`

Suite à l'application de cet opérateur, on doit vérifier comment la carte de processus intégrée traite le fait que tout scénario doive appartenir à un cas d'utilisation. La carte intégrée montre que selon le chemin choisi dans la carte nous pouvons obtenir deux solutions différentes. En écrivant les scénarios suivant les directives de CREWS-*L'Ecritoire* nous obtenons un ensemble de scénarios conceptualisés, tandis que suivant les directives proposées par le composant OOSE nous obtenons un ensemble de cas d'utilisation. La solution possible pour arriver au même résultat dans tous les cas est d'ajouter une nouvelle section permettant de regrouper les scénarios alternatifs en cas d'utilisation. Par conséquent, on choisit la section *<Construire un modèle de processus intégré, Construire un modèle de processus intégré, Stratégie d'addition>* et on applique l'opérateur `AJOUTER_SECTION` pour connecter les intentions *"Conceptualiser un scénario"* et *"Conceptualiser un cas d'utilisation"* avec une nouvelle stratégie appelée *"stratégie d'intégration"*. Cette section doit proposer une directive permettant d'intégrer les scénarios obtenus en utilisant le processus de CREWS-*L'Ecritoire* en des cas d'utilisation équivalent aux cas d'utilisation obtenus en utilisant OOSE. Puisque la directive

correspondante n'existe pas encore, elle doit être définie lors de l'application de l'opérateur `AJOUTER_SECTION` (Situation 7 au Tableau 6).

De plus, la section <*Conceptualiser un scénario, Arrêter, Stratégie de complétude*> doit être éliminée de la carte intégrée, ce qui permet d'obliger l'ingénieur d'applications à intégrer toujours les scénarios dans des cas d'utilisation. On sélectionne alors la section <*Construire un modèle de processus intégré, Construire un modèle de processus intégré, Stratégie de suppression*> dans la carte d'assemblage et on supprime cette section à l'aide de l'opérateur `SUPPRIMER_SECTION`. La directive DSI associée à l'intention *"Conceptualiser un scénario"* de la carte intégrée doit aussi être modifiée : la possibilité de progresser vers l'intention *"Arrêter"* doit être supprimée et la possibilité de progresser vers l'intention *"Conceptualiser un cas d'utilisation"* doit être ajoutée (Situation 8 au Tableau 6).

De la même manière que l'on a procédé avec la relation "OU", on transforme la relation *"ET"* entre les *"Fragments de besoin"* dans le modèle de produit de CREWS-*L'Ecritoire* en un nouveau concept appelé *"Famille des cas d'utilisation"* avec l'aide de l'opérateur `OBJECTIFIER_LIEN` (Situation 9 du Tableau 6).

On continue maintenant l'intégration des modèles de produit en sélectionnant la section <*Adapter les modèles de produit, Construire un modèle de processus intégré, Stratégie de spécialisation*> qui nous permet de définir un lien de spécialisation entre le concept *"Modèle des cas d'utilisation"* et le concept *"Famille des cas d'utilisation"*. A cause de leur différence structurelle nous décidons de ne pas les fusionner mais plutôt de les faire cohabiter dans le même modèle en définissant un lien de spécialisation entre les deux concepts (Situation 10 du Tableau 6). L'application de l'opérateur `SPECIALISER` permet d'ajouter un tel lien dans le modèle de produit intégré.

La fusion précédente des stratégies *"Découvrir un but"* dans la carte intégrée implique l'intégration de la notion *"But"* des deux modèles de produit. La notion du *"But"* dans le modèle de produit de CREWS-*L'Ecritoire* représente l'objectif d'un cas d'utilisation comme le fait la propriété *"Objectif"* du concept *"Cas d'utilisation"* dans le modèle de produit de OOSE. L'application de l'opérateur `OBJECTIFIER_PROPRIETE` sur la propriété *"Objectif"* permet d'unifier la terminologie des deux modèles de produits (Situation 11 du Tableau 6).

Puisque les structures du concept *"But"* et du nouveau concept *"Objectif"* sont différentes, on décide de garder les deux concepts en faisant le lien entre eux par le moyen de la généralisation. Pour cela, il faut renommer le concept *"But"* en *"But formel"* dans le modèle de produit de CREWS-*L'Ecritoire* et le concept *"Objectif"* en *"But informel" dans celui de* OOSE en sélectionnant deux fois de suite la section <*Démarrer, Adapter les modèles de produit, Stratégie d' unification*> (Situations 12 et 13 du Tableau 6). Et finalement, on applique l'opérateur `GENERALISER_CONCEPT` sur les concepts *"But formel"* et *"But informel"* pour définir un nouveau concept appelé *"But"* qui permet de faire le lien entre ces concepts (Situation 14 du Tableau 6). Le résultat de l'application des situations de 11 à 14 est illustré à la Figure 136.

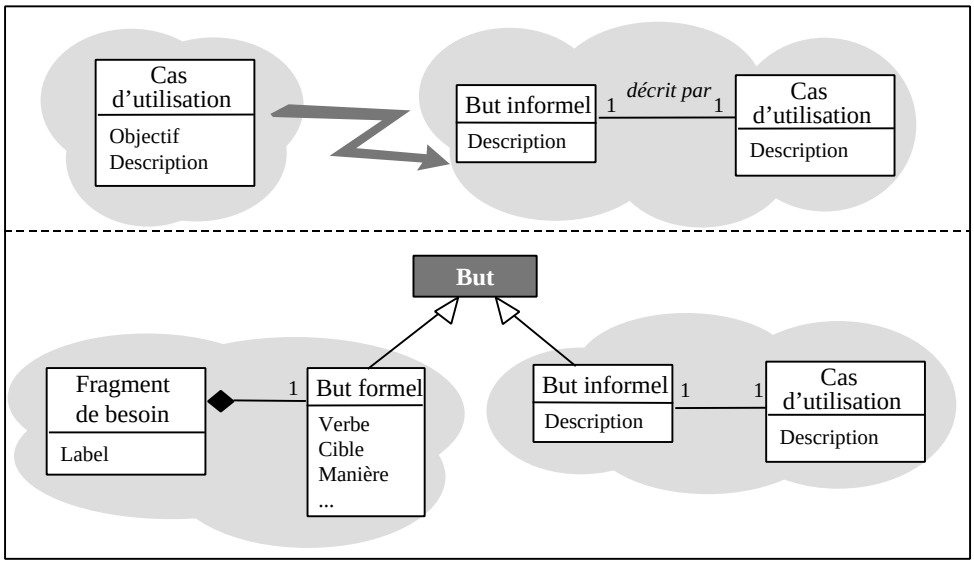


Figure 136 : Application des opérateurs `OBJECTIFIER_PROPRIETE` et `GENERALISER`

La section <*Construire un modèle de processus intégré, Construire un modèle de processus intégré, Stratégie de généralisation*> dans la carte d'assemblage de la Figure 127 permet de progresser vers l'intégration des concepts de *"scénario"*.

Les deux modèles de produit contiennent le concept *"scénario"*. La différence structurelle des deux concepts ne permet pas de les fusionner. Le *"scénario"* de CREWS-*L'Ecritoire* est composé d'un état initial, d'un état final et d'un ensemble d'actions, tandis que le *"scénario"* de OOSE est une description informelle non structurée. On décide de les renommer en *"scénario informel"* (dans OOSE) et *"scénario semi formel"* (dans CREWS-*L'Ecritoire*) en appliquant les opérateurs `RENOMMER_CONCEPT` (Situations 15 et 16 du Tableau 6), et de les généraliser ensuite en un nouveau concept *"scénario"* à l'aide de l'opérateur `GENERALISER` (Situation 17 du Tableau 6).

On poursuit l'affinement du modèle de processus intégré en se servant de la carte. Le modèle de processus obtenu offre deux possibilités pour écrire les scénarios : en suivant les directives d'écriture et de conceptualisation proposées par le composant CREWS-*L'Ecritoire* ou en suivant les directives proposées par le composant OOSE. Les directives de CREWS-*L'Ecritoire* sont plus riches et plus complètes que les directives de OOSE. De plus, les scénarios obtenus en utilisant les premières directives sont semi-formels et peuvent être validés par un outil tandis que suivant les deuxièmes directives nous obtenons des scénarios informels. Pour pouvoir appliquer les directives de recherche des nouveaux buts à partir des scénarios nous avons besoin des scénarios ayant la structure définie par CREWS-*L'Ecritoire*. Nous avons deux solutions possibles : la première consiste à éliminer la stratégie *"cas normal d'abord"* proposée par OOSE et interdire ainsi l'écriture des scénarios informels, la seconde consiste à ajouter une nouvelle stratégie permettant de transformer les scénarios écrits selon la méthode OOSE en scénarios ayant la structure de CREWS-*L'Ecritoire*. Dans le premier cas, les modifications correspondantes sur le modèle de produit entraînent la suppression du concept *"scénario informel"* défini dans OOSE. Dans le second cas, les deux concepts *"scénario"* doivent cohabiter dans le schéma du modèle intégré. La première solution diminue donc la flexibilité du processus intégré tandis que la seconde l'augmente. C'est donc cette dernière qui est retenue et l'on

ajoute une nouvelle section <*Conceptualiser un cas d'utilisation, Conceptualiser un cas d'utilisation, Stratégie de transformation*> dans la carte intégrée (Situation 18 du Tableau 6). Cette addition consiste à créer une nouvelle DRI qui permettra de transformer les scénarios informels de OOSE en scénarios semi-formels de CREWS-*L'Ecritoire*.

En validant le modèle de produit intégré on s'aperçoit que le lien entre les concepts "Cas d'utilisation" et "Scénario informel" n'a plus de raison d'exister. On décide alors de le supprimer à l'aide de l'opérateur `SUPPRIMER_LIEN` (Situation 19 du Tableau 6).

Finalement, on décide que les modèles de produit et de processus sont bien intégrés et on exécute les DRI 17 et 18 qui permettent de valider la complétude des modèles obtenus avec les règles de cohérence et de complétude présentées au Chapitre 5.

Les figures 17 et 18 illustrent les modèles de produit et processus intégrés.

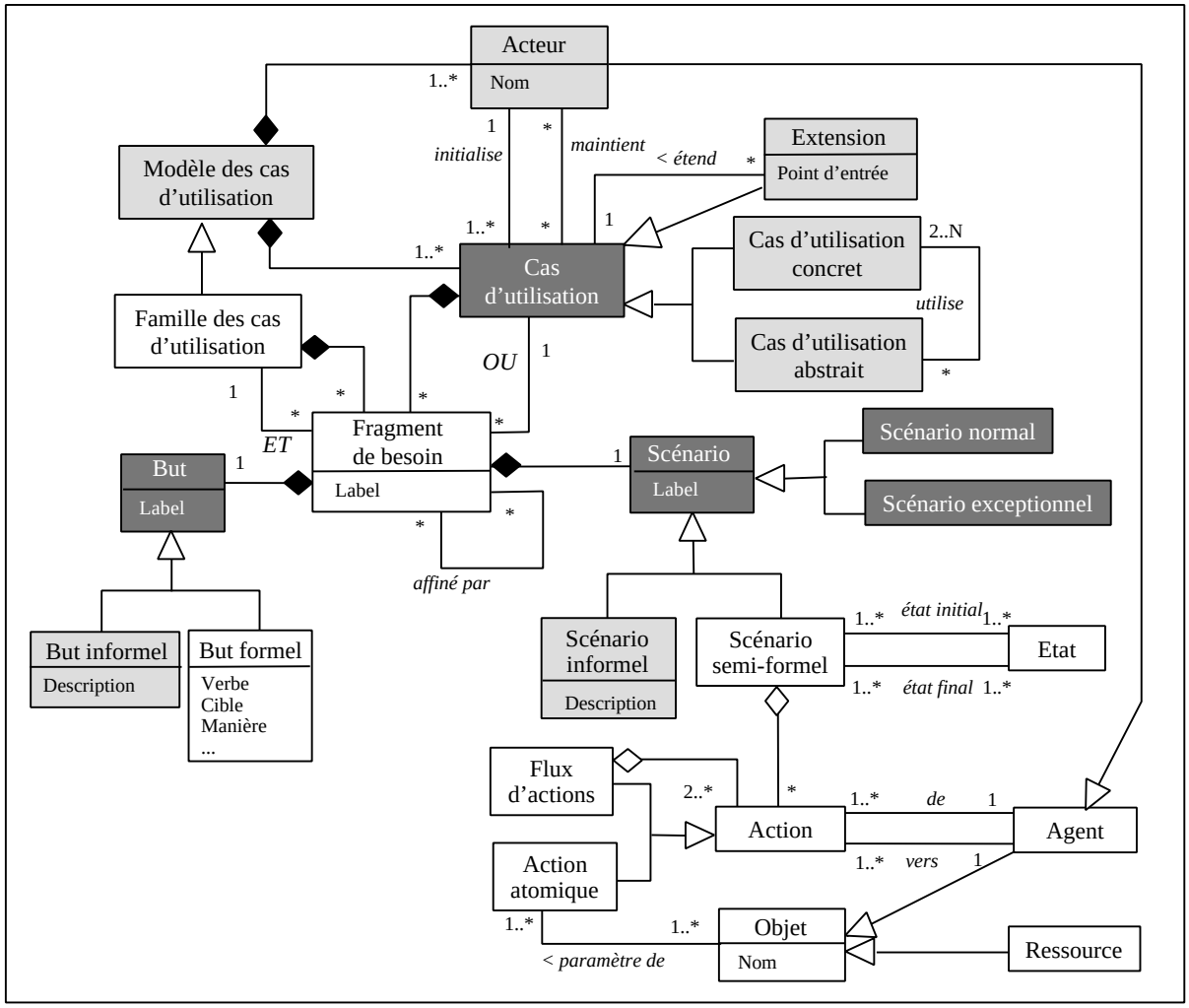


Figure 137 : Modèle de produit intégré

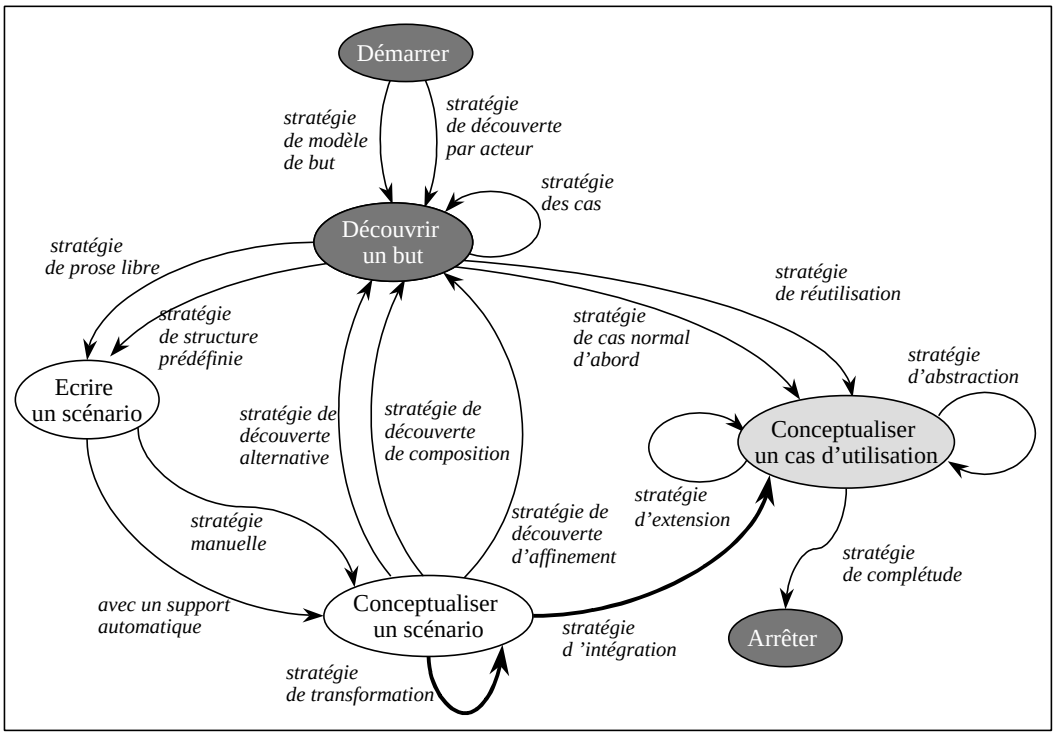


Figure 138 : Modèle de processus intégré

Le résumé de la démarche que l'on a appliquée est présenté au Tableau 6 en faisant apparaître la section sélectionnée dans la carte d'assemblage présentée à la Figure 127, les opérateurs d'assemblage exécutés par la directive associée et les autres actions nécessaires à l'application de celle-ci. Les lignes grises de ce tableau concernent l'intégration des modèles de produit et les lignes blanches correspondent à l'intégration des modèles de processus. La Figure 139 montre les sections qui ont été choisies lors de l'assemblage des composants OOSE et CREWS-*L'Ecritoire et dans* quel ordre.

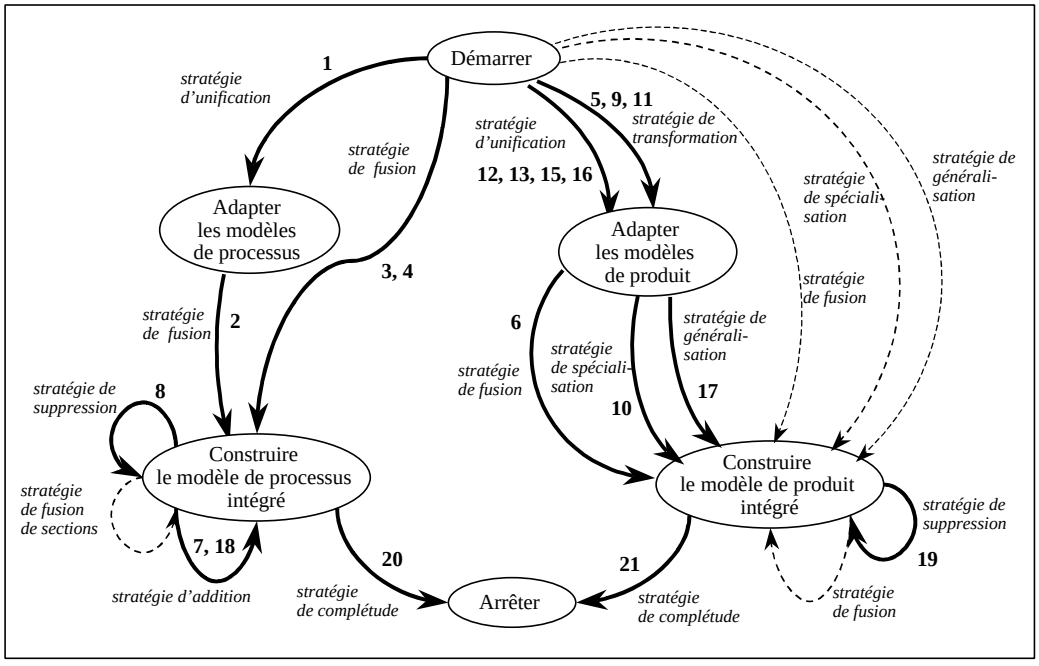


Figure 139 : La démarche utilisée pour l'assemblage des composants OOSE et CREWS-*L'Ecritoire*

N°	Section sélectionnée	Opérateur d'assemblage	Actions
1	<Démarrer, Adapter les modèles de processus, Stratégie d'unification>	`RENOMMER_INTENTION(OOSE, Découvrir un cas d'utilisation, Découvrir un but)`	Identification d'un couple d'intentions similaires : 1. *Découvrir un cas d'utilisation* dans la carte de OOSE 2. *Découvrir un but* dans la carte de CREWS-*L'Ecritoire* Décision de renommer l'intention *Découvrir un cas d'utilisation.* Changement du nom de l'intention : la nouvelle cible – *un but.* Modification de la signature de la directive DSI associée à l'intention renommée: *<(Objectif d'un cas d'utilisation), Progresser de Découvrir un cas d'utilisation>* par *<(But), Progresser de Découvrir un but>*.
2	<Adapter les modèles de processus, Construire un modèle de processus intégré, Stratégie de fusion>	`FUSIONNER_INTENTION (OOSE, Découvrir un but, CREWS-L'Ecritoire, Découvrir un but)`	Fusion des intentions *Découvrir un but*. Construction d'une nouvelle DSI (voir la Figure 133).
3	<Démarrer, Construire un modèle de processus intégré, Stratégie de fusion>	`FUSIONNER_INTENTION (OOSE, Démarrer, CREWS-L'Ecritoire, Démarrer)`	Fusion des deux intentions *Démarrer*. Construction d'une nouvelle DSS de l'intention *Démarrer* (voir la Figure 134).
4	<Démarrer, Construire un modèle de processus intégré, Stratégie de fusion>	`FUSIONNER_INTENTION (OOSE, Arrêter, CREWS-L'Ecritoire, Arrêter)`	Fusion des deux intentions *Arrêter.*
5	<Démarrer, Adapter les modèles de produit, Stratégie de transformation>	`OBJECTIFIER_LIEN (CREWS-L'Ecritoire, OU(Fragment de besoin, Fragment de besoin), Cas d'utilisation)`	Création d'un nouveau concept *Cas d'utilisation* dans le modèle de produit de CREWS-*L'Ecritoire*. Création des deux liens entre le concept *Cas d'utilisation* et le concept *Fragment de besoin.* Suppression du lien *OU* entre le concepts *Fragment de besoin.*
6	<Adapter les modèles de produit, Construire un modèle de produit intégré, Stratégie de	`FUSIONNER_CONCEPT (OOSE, Cas d'utilisation, CREWS-L'Ecritoire, Cas`	Fusion des deux concepts *Cas d'utilisation* en un concept intégré.

	fusion>	d'utilisation)	
7	<Construire un modèle de processus intégré, Construire un modèle de processus intégré, Stratégie d'addition>	AJOUTER_SECTION(MPdI[5], <Conceptualiser un scénario, Conceptualiser un cas d'utilisation, stratégie d'intégration>)	Addition d'une nouvelle section : <*Conceptualiser un scénario, Conceptualiser un cas d'utilisation, Stratégie d'intégration*>. Création d'une DRI : <*({Scénario}), Intégrer les scénarios*>. Modification de la DSI de l'intention *Conceptualiser un scénario*.
8	<Construire un modèle de processus intégré, Construire un modèle de processus intégré, Stratégie de suppression>	SUPPRIMER_SECTION(MPdI, <Conceptualiser un scénario, Arrêter, Stratégie de complétude>)	Suppression de la section <Conceptualiser un scénario, Arrêter, Stratégie de complétude> Suppression de la DRI : <*({Scénario}), Arrêter avec la stratégie de complétude*>. Modification de la DSI associée à l'intention *Conceptualiser un scénario*.
9	<Démarrer, Adapter les modèles de produit, stratégie de transformation>	OBJECTIFIER_LIEN (CREWS-*L'Ecritoire*, ET(Fragment de besoin, Fragment de besoin), Famille des cas d'utilisation)	Création d'un nouveau concept *Famille de cas d'utilisation*. Création des deux liens entre le concept *Famille de cas d'utilisation* et le concept *Fragment de besoin*. Suppression du lien *ET* entre les concepts *Fragment de besoin*.
10	<Adapter les modèles de produit, Construire un modèle de produit intégré, Stratégie de spécialisation>	SPECIALISER (MPcI[6], Famille des cas d'utilisation, Modèle des cas d'utilisation)	Création d'un lien de spécialisation entre le concept *Famille des cas d'utilisation* et le concept *Modèle des cas d'utilisation*.
11	<Démarrer, Adapter les modèles de produit, stratégie de transformation>	OBJECTIFIER_PROPRIETE (OOSE, Objectif(Cas d'utilisation), Objectif)	Création d'un nouveau concept *Objectif* dans OOSE. Créer un nouveau lien entre le concept *Objectif* et le concept *Cas d'utilisation*.

[5] MpdI : Modèle de produit intégré

[6] MpcI : Modèle de processus intégré

			Suppression de la propriété *Objectif* du concept *Cas d'utilisation*.
12	<Démarrer, Adapter les modèles de produit, Stratégie d'unification>	`RENOMMER_CONCEPT(OOSE, Objectif, But informel)`	Changement du nom du concept *Objectif* en *But informel* dans OOSE.
13	<Démarrer, Adapter les modèles de produit, Stratégie d'unification>	`RENOMMER_CONCEPT(CREWS-L'Ecritoire, But, But formel)`	Changement du nom du concept *But* en *But formel* dans CREWS-*L'Ecritoire.*
14	<Adapter les modèles de produit, Construire un modèle de produit intégré, Stratégie de généralisation>	`GENERALISER_CONCEPT(OOSE, But informel, CREWS-L'Ecritoire, But formel, MPdI, But)`	Création d'un nouveau concept *But* dans le MPdI. Création d'un lien de généralisation entre le concept *But informel* et le concept *But*. Création d'un lien de généralisation entre le concept *But formel* et le concept *But*.
15	<Démarrer, Adapter les modèles de produit, Stratégie d'unification>	`RENOMMER_CONCEPT(CREWS-L'Ecritoire, Scénario, Scénario semi formel)`	Changement de nom du concept *Scénario* par *Scénario semi-formel* dans le modèle de produit de CREWS-*L'Ecritoire*.
16	<Démarrer, Adapter les modèles de produit, Stratégie d'unification>	`RENOMMER_CONCEPT(OOSE, Scénario, Scénario informel)`	Changement de nom du concept *Scénario* en *Scénario informel* dans le modèle de produit de OOSE.
17	<Adapter les modèles de produit, Construire un modèle de produit intégré, Stratégie de généralisation>	`GENERALISER(MPcI, Scénario semi formel, Scénario informel, Scénario)`	Création d'un nouveau concept *Scénario*. Création d'un nouveau lien de généralisation entre le concept. *Scénario informel* et le concept *Scénario*. Création d'un nouveau lien de généralisation entre le concept. *Scénario semi-formel* et le concept *Scénario*.
18	<Construire un modèle de processus intégré, Construire un modèle de processus intégré, Stratégie d'addition>	`AJOUTER_SECTION(MPdI, <Conceptualiser un scénario, Conceptualiser un scénario, stratégie de transformation>)`	Création d'une DRI : *<(Scénario informel), Transformer un scénario informel en scénario semi-formel>*. Modification de la DSI de l'intention *Conceptualiser un scénario*.
19	<Construire un modèle de produit intégré, Construire un modèle de produit intégré, Stratégie de suppression>	`SUPPRIMER_LIEN(MPcI, nomL(Cas d'utilisation, Scénario informel)`	Suppression du lien entre les concepts *Cas d'utilisation* et *Scénario informel* dans le modèle de produit intégré.

20	<Construire un modèle de processus intégré, Arrêter, Stratégie de complétude>		Vérification de la complétude des modèles de produit et de processus intégrés.
21	<Construire un modèle de produit intégré, Arrêter, Stratégie de complétude>		Vérification de la complétude des modèles de produit et de processus intégrés.

Tableau 6 : Résumé de la démarche appliquée lors de l'assemblage des composants OOSE et CREWS-*L'Ecritoire*

3.5 Assemblage des composants par association

3.5.1 Descripteur

Situation de réutilisation

Domaine d'application : Ingénierie de méthodes

Activité : Assemblage de composants de méthode

Intention de réutilisation

Associer des composants de méthodes

Objectif

Ce composant d'assemblage est destiné à l'assemblage de composants de méthodes ayant des démarches complémentaires dans le processus d'ingénierie d'un système d'information. Les cartes de processus de tels composants n'ont pas d'intentions communes et leur modèles de produit n'ont pas d'élément commun dans la plupart des cas.

Type : Atomique

3.5.2 Composant

Identifiant : CA3-2

Signature

Situation : Deux composants de méthode Co1 et Co2

Intention : Assembler les deux composants Co1 et Co2 de méthodes avec la stratégie d'association

Directive

Type de directive : Stratégique

Représentation formelle : Le processus d'assemblage est représenté par une carte à la Figure 140. Les directives associées à la carte sont répertoriées dans le Tableau 7.

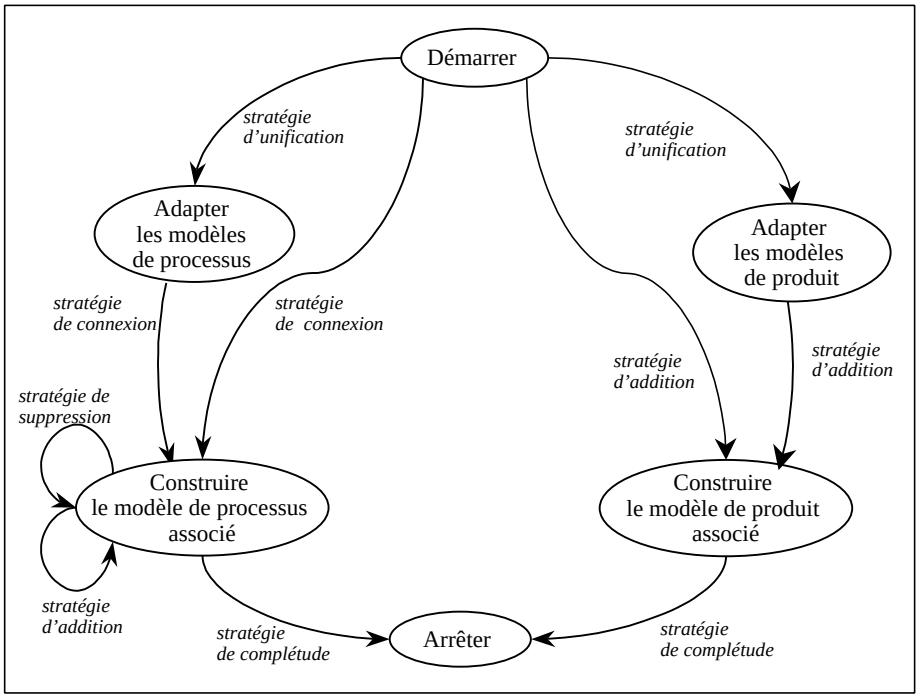
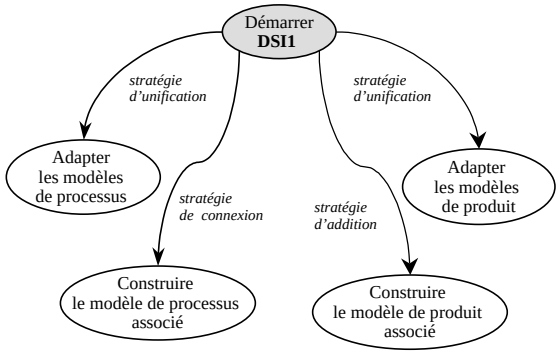
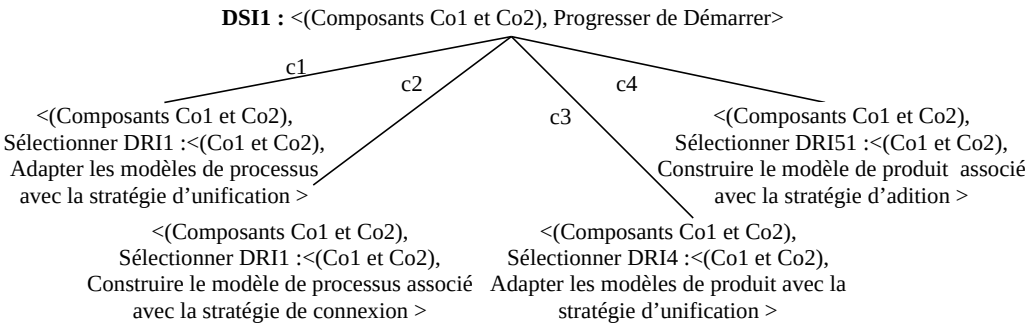
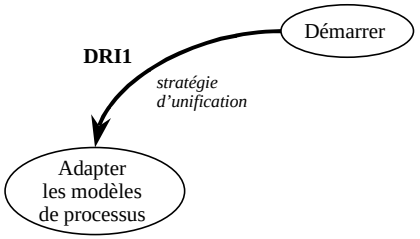
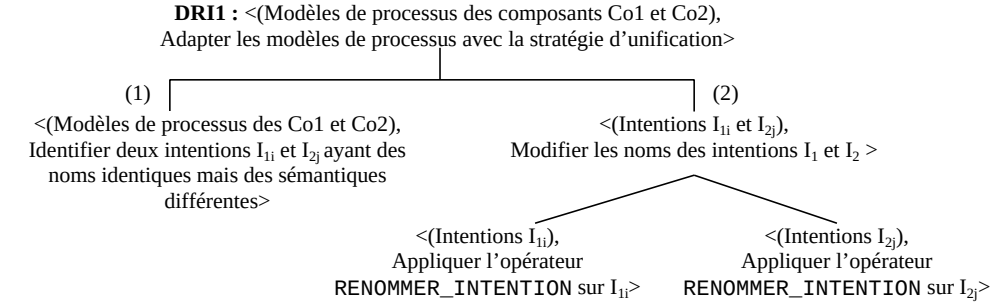
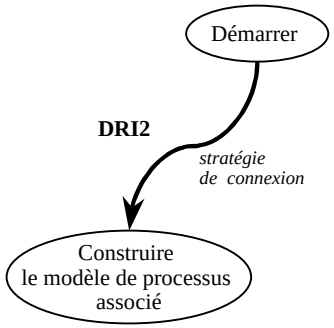
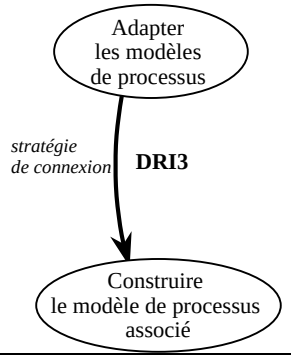
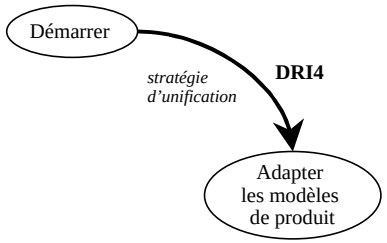
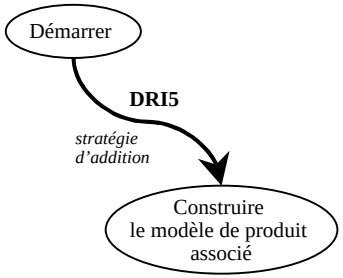


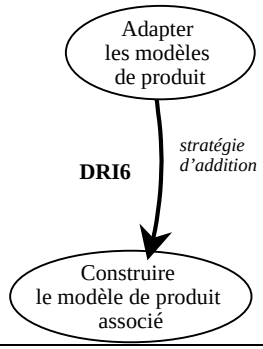
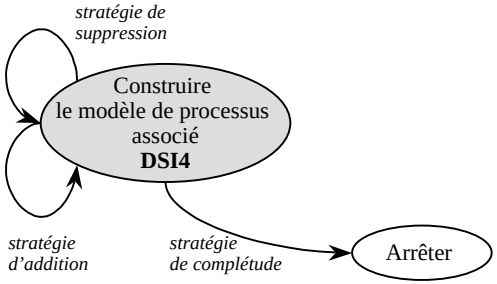
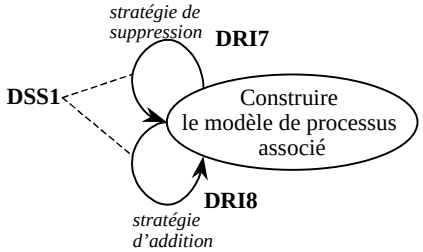
Figure 140 : Processus d'assemblage par association

<table>
<tr><th>N°</th><th>Situation dans la carte de processus d'assemblage</th><th>Directive correspondant à la situation</th></tr>
<tr><td>1</td><td> 
</td><td> 

Critères de choix :
c1 : a1 ET a3 ; c2 : NON a1 ET a3 ; c3 : a2 ET a4 ; c4 : NON a2 ET a4.
Arguments :
a1 : Il existe des intentions ayant le même nom dans les modèles de processus des composants.
a2 : Il existe des concepts ayant le même nom dans les modèles de produit des composants.
a3 : L'ingénieur d'applications préfère commencer l'assemblage par l'association des modèles de processus.
a4 : L'ingénieur d'applications préfère commencer l'assemblage par l'association des modèles de produit.</td></tr>
<tr><td>2</td><td> 
</td><td> 
</td></tr>
</table>

<table>
<tr><td></td><td></td><td>(1) Appliquer la mesure de similarité des carte ASI, elle doit être égale à 0.
(2) Appliquer l’opérateur <code>RENOMMER_INTENTION</code> sur une des deux intentions.</td></tr>
<tr><td>3</td><td> 
</td><td>DRI2 : <(Modèles de processus des composants Co1 et Co2),
Construire le modèle de processus associé avec la stratégie de connexion>
(1) <(Composants Co1 et Co2), Identifier le composant source Cos et le composant cible Coc parmi Co1 et Co2 >
(2) <(Modèle de processus de Cos), Identifier l’intention source Is de la connexion des composants >
(3) <(Modèles de processus de Cos et Coc, Is), Appliquer l’opérateur FUSIONNER_INTENTION sur Is et l’intention <i>Démarrer</i> de Coc>
(1) Le composant source ici est celui qui construit le concept utilisé par le deuxième composant (cible) en tant que produit source.
(2) L’intention source ici est une des intentions du composant source, celle qui construit le produit nécessaire à l’exécution du composant cible.</td></tr>
<tr><td>4</td><td> 
</td><td>DRI3 : <(Modèles de processus des composants Co1 et Co2), Construire un modèle de processus associé avec la stratégie de connexion>
Cette directive est identique à la DRI2.</td></tr>
</table>

5		**DRI4 :** <(Modèles de produit des composants Co1 et Co2), Adapter les modèles de produit avec la stratégie d'unification> (1) <(Modèles de produit des Co1 et Co2), Identifier deux concepts C_{1i} et C_{2j} ayant le même nom mais les sémantiques différentes> (2) <(Concepts C_{1i} et C_{2j}), Unifier les noms des concepts C_{1i} et C_{2j} > <(Concept C_{1i}), Appliquer l'opérateur `RENOMMER_CONCEPT` sur C_{1i} > <(Concept C_{2j}), Appliquer l'opérateur `RENOMMER_CONCEPT` sur C_{2j} > (1) Mesurer la similarité sémantique des concepts C_{1i} et C_{2j} avec la mesure AN. Mesurer la similarité structurelle des concepts C_{1i} et C_{2j} avec la mesure ASG. (2) Appliquer l'opérateur `RENOMMER_CONCETP` sur l'un des deux concepts si leurs noms sont identiques mais les sémantiques différentes.
6		**DRI5 :** <(Modèles de produit des composants Co1 et Co2), Construire le modèle de produit associé avec la stratégie d'addition> (1) <(Modèles de produit des Co1 et Co2), Identifier deux concepts connecteurs $C_{1.i}$ et C_{2j}> (2) <(Concepts $C_{1.i}$ et C_{2j}), Connecter les concepts $C_{1.i}$ et C_{2j} > c1 <(Concepts $C_{1.i}$ et C_{2j}), Appliquer l'opérateur `AJOUTER_LIEN` entre les $C_{1.i}$ et C_{2j}> c2 <(Concepts $C_{1.i}$ et C_{2j}), Appliquer l'opérateur `AJOUTER_CONCEPT` pour connecter $C_{1.i}$ et C_{2j}> (1) Les concepts connecteurs ici sont ceux qui servent de passage d'un modèle de produit vers un autre. (2) Il y a deux possibilités pour faire la connexion entre ces concepts connecteurs : (2.1) En ajoutant un lien entre ces concepts si c'est possible (C_{1i}), (2.2) En ajoutant un concept de passage d'un concept connecteur vers un autre, si le premier cas n'est pas possible (C_{2j}).

7		**DRI6 :** <(Modèles de produit des composants Co1 et Co2), Construire le modèle de produit associé avec la stratégie d'addition> Cette directive est identique à la DRI5.
8		**DSI4 :** <(Modèle de processus associé MPcA), Progresser de Construire le modèle de processus associé> c1 : <(Modèle de processus associé MPcA), Sélectionner DSS1 :<(MPcA) Progresser vers Construire le modèle de processus associé>> c2 : <(Modèle de processus associé MPcA), Sélectionner DRI9 :<(Composant associé), Arrêter le processus d'assemblage avec la stratégie de complétude> c1 : Il est nécessaire d'affiner le modèle de processus associé. c2 : L'ingénieur d'applications décide que la construction du modèle de processus associé est terminée.
9		**DSS1 :** <(Modèle de processus associé MPcA), Progresser vers Construire le modèle de processus associé> c1 : <(Modèle de processus associé MPcA), Sélectionner DRI7 :<(MPcA), Construire le modèle de processus associé avec la stratégie de suppression>> c2 : <(Modèle de processus associé MPcA), Sélectionner DRI8 :<(MPcA), Construire le modèle de processus associé avec la stratégie d'addition>> c1 : Il existe une section dans la carte intégrée qui n'est plus pertinente dans le processus associé. c2 : Si le modèle de produit intégré permet la cohabitation de deux concepts ayant la même sémantique mais des structures différentes, il est nécessaire d'ajouter une section dans la carte intégrée permettant de transformer une instance d'un concept en une instance de l'autre concept. **DRI7 :** <(Modèle de processus associé), Construire un modèle de processus associé avec la stratégie de suppression>

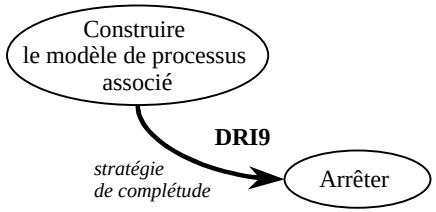
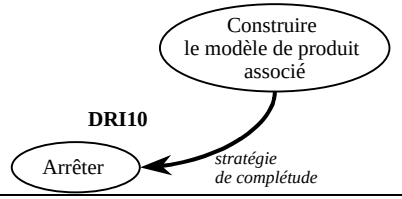
		Appliquer l'opérateur SUPPRIMER_SECTION sur la section identifiée. **DRI8 :** <(Modèle de processus associé), Construire un modèle de processus associé avec la stratégie d'addition> Appliquer l'opérateur AJOUTER_SECTION entre les intentions identifiées.
10		**DRI9** :<(Composant associé), Arrêter le processus d'assemblage avec la stratégie de complétude> <(Modèle de processus associé MPcA), Valider le MPcA> <(Modèle de produit associé MPdA), Valider le MPdA > Valider le modèle de processus et le modèle de produit associés en appliquant les règles de cohérence et de complétude de modèle de processus.
11		La DRI10 est identique à la DRI19.

Tableau 7 : Les directives associées à la carte d'assemblage par association

Description

Le processus d'assemblage par association consiste à déterminer l'ordre dans lequel les composant se succèdent. Comme dans le cas de l'assemblage par intégration le processus doit commencer par l'unification de la terminologie des composants si nécessaire (en renommant certains concepts ou intentions).

Le processus d'assemblage par association est beaucoup plus simple que celui par intégration. Il se limite à l'addition d'un lien ou d'un concept de connexion afin d'assembler deux modèles de produit et c'est seulement dans des cas exceptionnels que l'on peut être amené à ajouter plusieurs liens ou concepts de connexion. En ce qui concerne l'assemblage des modèles de processus, il consiste à déterminer lequel des deux composants doit être exécuté en premier pour construire un produit qui va être ensuite utilisé par le deuxième composant.

Situation 1 : Comme dans le cas de l'assemblage par intégration il est possible de démarrer le processus d'assemblage par l'association des modèles de processus et associer ensuite des modèles de produit ou bien commencer l'assemblage par l'association des modèles de produit puis associer les modèles de processus. Dans les deux cas il faut commencer le processus d'assemblage par l'adaptation des modèles correspondants.

Situation 2 : Il est parfois nécessaire d'adapter d'abord les cartes des composants afin de pouvoir les assembler. Dans ce cas d'assemblage les cartes des composants ne doivent pas avoir des intentions similaires puisque cela reviendrait à un assemblage par fusion. Par conséquent, les cartes ne doivent pas comporter des intentions ayant des noms identiques. S'il existe deux intentions des cartes différentes ayant le même nom, une des deux intentions doit être renommée pour la différencier.

La DRI1 associée à la section <*Démarrer, Adapter les modèles de processus, Stratégie d'unification*> propose tout d'abord d'identifier un couple d'intentions de deux cartes initiales ayant des noms identiques mais des sémantiques différentes et d'en renommer une en appliquant l'opérateur `RENOMMER_INTENTION`. Le coefficient de l'affinité sémantique de ces deux intentions doit être inférieur à 0.5 (ASI au Chapitre 5).

Situation 3 : Si les deux cartes initiales n'ont aucune intention de même nom, il est possible de passer directement à l'assemblage des cartes de composants sans passer par l'étape d'adaptation.

La DRI2 associée à la section <*Démarrer, Construire le modèle de processus intégré, Stratégie de connexion*> propose tout d'abord de définir l'ordre dans lequel les composants doivent être exécutés car dans ce cas d'assemblage, le deuxième composant utilise comme produit source le produit qui est le résultat d'exécution du premier composant. Par conséquent, dans la carte du premier composant il faut identifier l'intention qui permet de construire le produit nécessaire pour démarrer le deuxième. Ensuite, il faut fusionner cette intention avec l'intention "Démarrer" de la deuxième carte à l'aide de l'opérateur `FUSIONNER_INTENTION` (Chapitre 5). C'est un cas exceptionnel de fusion des intentions car les deux intentions n'ont pas le même nom. Le résultat de la fusion est une intention qui conserve le nom de la première.

Situation 4 : La DRI3 associée à la section <*Adapter les modèles de processus, Construire le modèle de processus intégré, Stratégie de connexion*> est la même que DRI2 mais appliquée sur les modèles de processus adaptés au préalable.

Situation 5 : La construction du modèle de produit intégré peut également être directe ou précédée par l'étape d'adaptation. Contrairement au cas d'assemblage par intégration (CA3-2), celui par association ne nécessite pas des transformations des modèles de produit telles que l'objectification des liens ou des propriétés. Ce cas se limite aux modifications des noms des concepts si nécessaire.

Puisque les deux modèles de produit n'ont aucun élément commun ils ne doivent pas non plus avoir des concepts ayant des noms identiques. La DRI4 associée à la section <*Démarrer, Adapter les modèles de produit, Stratégie d'unification*> permet de résoudre ce problème en proposant de renommer un des concepts. Tout d'abord, les mesures de similarité AN (affinité des noms) et ASG (affinité structurelle globale) (Chapitre 5) doivent être calculées afin de pouvoir comparer les deux concepts sélectionnés. Ensuite, si ces calculs montrent que les concepts ont les mêmes noms mais des sémantiques différentes, l'opérateur `RENOMMER_CONCEPT` doit être appliqué sur l'un des concepts pour différencier leurs noms.

Situation 6 : La construction du modèle de produit intégré est également possible sans passer par l'étape d'adaptation à condition qu'il n'y ait pas de concepts ayant les mêmes noms dès le départ. La DRI5 associée à la section <*Démarrer, Construire un modèle de produit intégré, Stratégie d'addition*> propose de connecter les deux modèles de produit par un nouveau lien ou même en introduisant un nouveau concept si nécessaire. Elle utilise les opérateurs `AJOUTER_LIEN` ou `AJOUTER_CONCEPT` suivant le cas.

Situation 7 : La DRI6 associée à la section <*Adapter les modèles de produit, Construire le modèle de produit intégré, Stratégie d'addition*> est identique à DRI5 mais appliquée sur les modèles de produit adaptés au préalable.

Situation 8 : Une fois que l'ingénieur d'applications a associé les deux cartes, la carte d'assemblage par association offre le choix entre deux possibilités :

- affiner le résultat obtenu (situation 9) ou
- décider que l'intégration des modèles de processus est terminée et vérifier la complétude du résultat obtenu (situation 10 du Tableau 7).

Situation 9 : La carte de processus d'assemblage (Figure 140) propose deux stratégies pour affiner le modèle de processus assemblé : la *stratégie de suppression* et la *stratégie d'addition*.

La DRI7 associée à la section <*Construire le modèle de processus intégré, Construire le modèle de processus intégré, Stratégie de suppression*> permet de supprimer les sections que l'ingénieur d'applications ne veut plus garder dans la carte d'assemblage. Par exemple, la section menant à l'arrêt du processus du premier composant peut être supprimée afin d'interdire la fin du processus d'ingénierie de l'assemblage après avoir exécuté la démarche du premier composant et sans avoir

exécuté la démarche du deuxième composant. La directive applique l'opérateur `SUPPRIMER_SECTION` sur les sections contestables.

La DRI18 associée à la section <*Construire le modèle de processus intégré, Construire le modèle de processus intégré, Stratégie d'addition*> propose d'ajouter une section de connexion entre deux intentions d'origine de cartes différentes si nécessaire ou bien une section permettant de transformer un produit généré par le premier composant pour le rendre compatible avec le produit d'entrée du second. La transformation du produit peut être la solution dans ce cas. Pour cela, la directive applique l'opérateur `AJOUTER_SECTION`.

Situation 10 - 11: Les DRI9 et 10 sont destinées à la validation du résultat obtenu. Elles vérifient la cohérence et la complétude du modèle de produit et du modèle de processus associés.

3.5.3 Exemple d'assemblage des composants par association

Pour illustrer l'application du composant CA3-2 nous avons choisi d'associer les composants des méthodes CREWS-*L'Ecritoire* et *Albert* [Heymans 98], [Dubois 98] qui représentent deux processus complémentaires dans la conception d'un système d'information. Le premier sert à découvrir des besoins du système et à les conceptualiser sous forme de buts et de scénarios tandis que le deuxième permet de valider ces besoins par les animations des scénarios écrits au préalable. Ceci veut dire que le premier composant produit des scénarios qui sont utilisés par le deuxième. Par conséquent, l'assemblage de ces composants est de type *par association.*

Le composant CREWS-*L'Ecritoire* est présenté en détail à la section 3.4.3 de ce chapitre. Avant de commencer l'assemblage de ces composants nous présentons d'abord le composant de méthode *Albert* dont le modèle de processus et le modèle de produit sont présentés à la Figure 141.

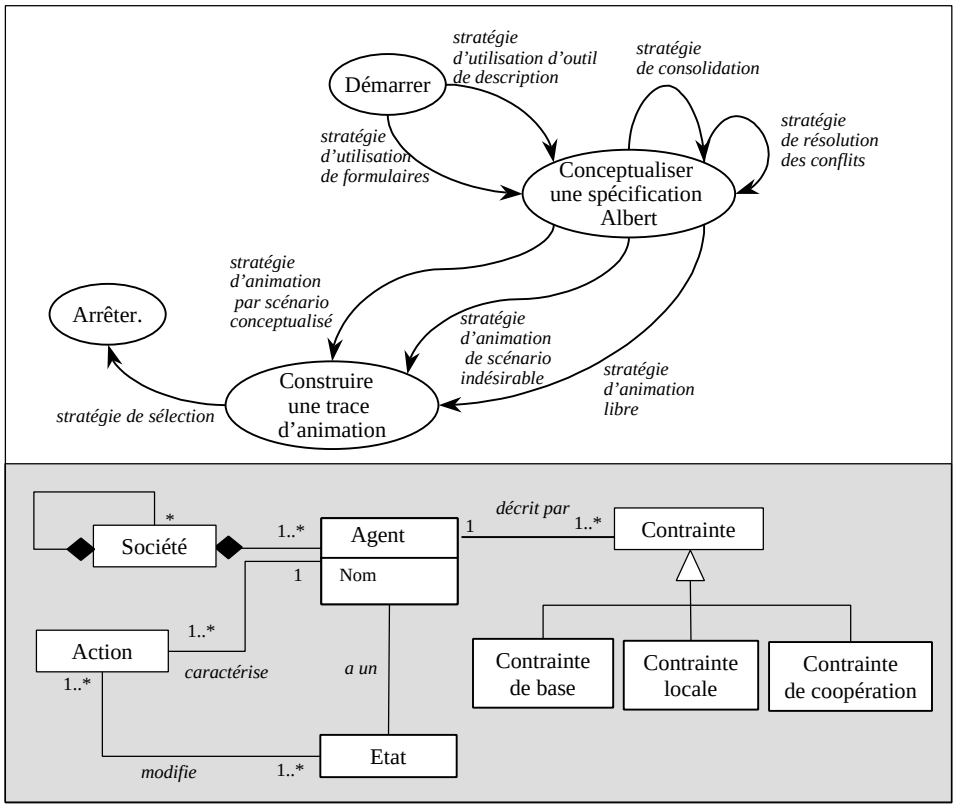


Figure 141 : Composant de méthode *Albert*

Comme le montre la Figure 141, ce composant aide à transformer un scénario en une spécification *Albert* à l'aide des formulaires prédéfinis et/ou en utilisant un outil de description. Ceci est représenté par deux stratégies de conceptualisation d'une spécification en langage Albert : la *stratégie d'utilisation de formulaire* et la *stratégie d'utilisation d'outil de description*. Ensuite, la *stratégie de résolution des conflits* et la *stratégie de consolidation* permettent d'analyser et de valider cette spécification. Une fois la spécification validée, elle est animée au moyen d'un outil appelé *Animateur* qui permet de visualiser l'exécution du scénario et de valider le besoin correspondant par le futur utilisateur. Le composant propose plusieurs stratégies d'animation : *l'animation par scénario conceptualisé*, *l'animation par scénario indésirable* et *l'animation libre*. Chaque cas construit une trace d'animation que l'ingénieur de développement et/ou l'utilisateur du système peut valider en acceptant ou en refusant le besoin validé lors de l'animation.

La partie grisée de la Figure 141 décrit le modèle de produit du composant *Albert*. Une spécification *Alber* est composée d'une hiérarchie d'*agents*. Les agents sont groupés en sociétés qui à leurs tours peuvent être groupées en sociétés encore plus large et ainsi de suite. Les *agents* d'une société interagissent entre eux afin de produire des services que le système est sensé offrir à ses utilisateurs. Chaque agent est caractérisé par un ensemble d'actions qui modifient ou maintiennent son état. Les *actions* sont exécutées par les *agents* afin d'accomplir leurs obligations exprimées en terme des *contraintes*.

3.5.3.1 Démarche utilisée pour l'association des composants CREWS-*L'Ecritoire et Albert*

Comme dans le cas précédent, on décide de démarrer l'assemblage des composants par l'association de leurs modèles de processus. Puisque les deux cartes correspondantes (Figure 141et Figure 131) n'ont aucune intention commune, on peut passer directement à la construction du nouveau modèle de processus sans passer par l'étape d'adaptation des cartes. On sélectionne alors la section <*Démarrer, Construire le modèle de processus associé, Stratégie de connexion*>. Suivant la DRI2 associée à cette section on identifie que l'intention "*Conceptualiser un scénario"* de CREWS-*L'Ecritoire* est une intention construisant le produit (le scénario) qui est le produit source pour le composant *Albert*. Ensuite on applique l'opérateur `FUSIONNER_INTENTION` sur l'intention "*Conceptualiser un scénario"* et l'intention "*Démarrer"* d'*Albert* en préservant le nom de la première intention. L'opérateur modifie la DSI de l'intention *"Conceptualiser un scénario"* en y ajoutant le choix de la DSS qui correspond et qui propose le choix entre les deux stratégies menant vers la construction d'une spécification *Albert*.

Le résultat de la validation par animation se résume à la décision d'accepter ou de refuser le but dont la réalisation a été validée lors de l'animation. On peut décider alors, qu'il est nécessaire de valider ainsi tous les buts découverts précédemment. Afin d'interdire la progression de l'intention *"Conceptualiser un scénario"* directement vers l'intention *"Arrêter"* sans passer par l'étape de validation dans la nouvelle carte associée on décide de supprimer la section <*Conceptualiser un scénario, Arrêter, Stratégie de complétude*>. Mais avant tout, pour garder ce processus de vérification de la complétude, on ajoute une autre section <*Conceptualiser un scénario, Conceptualiser un scénario, Stratégie de complétude*> qui préserve ce processus. Pour cela, on sélectionne tout d'abord la *stratégie d'addition* dans la carte d'assemblage par association (Figure 140) et on applique l'opérateur `AJOUTER_SECTION`, ensuite on sélectionne la *stratégie de suppression* et on applique l'opérateur `SUPPRIMER_SECTION` sur la section mentionnée plus haut. Le résultat d'association des deux cartes est illustré à la Figure 142**.**

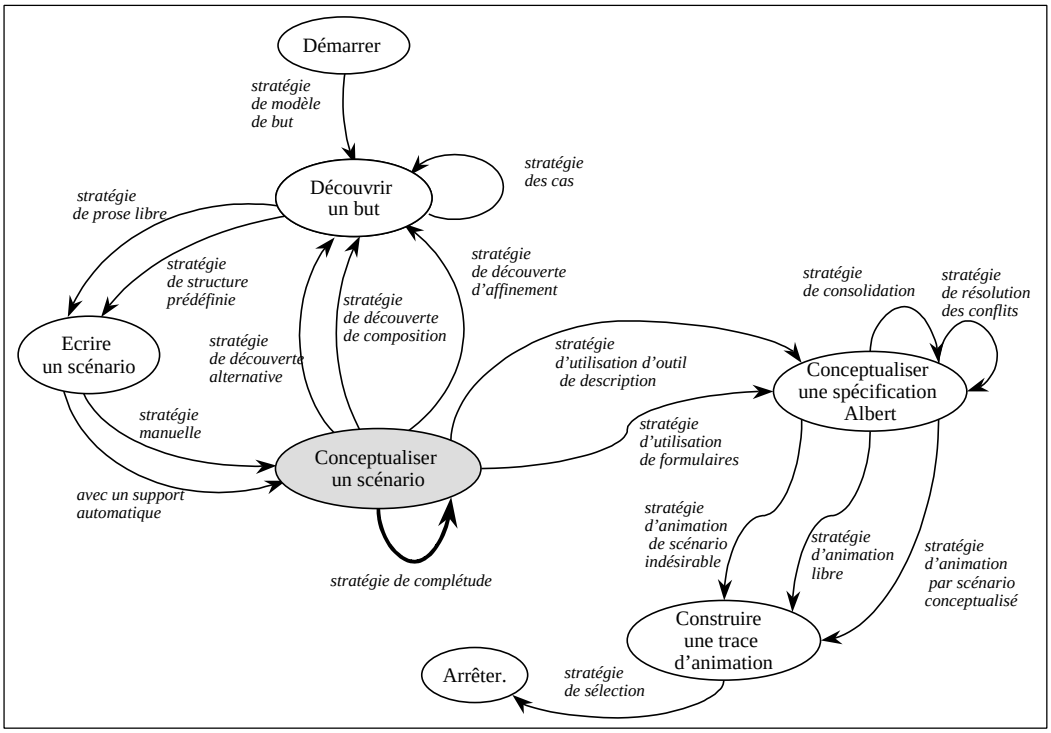


Figure 142 : Modèle de processus associé

Passons désormais à l'assemblage des modèles de produit correspondants. Le concept d'agent existe dans les deux modèles de produit. Même si la sémantique des deux concepts est similaire, leur structures sont complètement différentes. Par conséquent, il est nécessaire de renommer un des deux composants afin de commencer l'association des deux modèles de produits. On propose de renommer l'*agent* du composant *Albert* en l'appelant *Agent_Albert*. Le concept *Etat* pose le même problème, il figure dans les deux modèles de produit mais la sémantique respective des deux concepts n'est pas la même. Dans CREWS-*L'Ecritoire* il représente les conditions d'exécution d'un scénario tandis que dans *Albert* c'est un état d'un Agent. On propose alors de renommer ce concept en *Etat d'agent* dans le modèle de produit d'*Albert*. Finalement, un des deux concepts Action doit également être renommé car leurs structures sont trop différentes. Chaque fois on passe par la *stratégie de rénommage* et on applique l'opérateur `RENOMMER_CONCEPT`.

Maintenant, passons à l'association des deux modèles de produit. Pour ceci, on sélectionne la stratégie d'addition dans la carte d'assemblage par association (Figure 140) et on applique l'opérateur `AJOUTER_LIEN`.

Le résultat d'association des deux modèles de produit est illustré à la Figure 143.

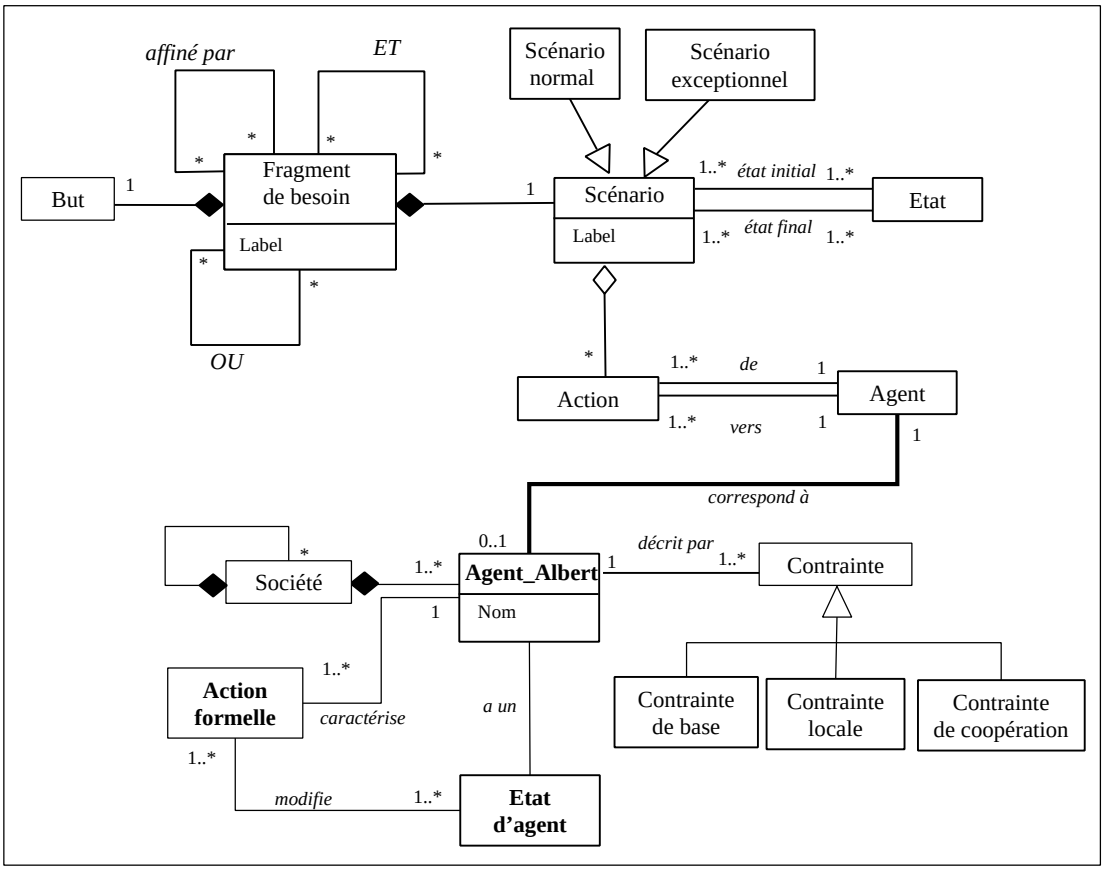


Figure 143 : Modèle de produit associé

Comme les présentations des composants proposés dans les sections précédentes celle-ci résume également la démarche appliquée dans le Tableau 8 qui fait apparaître les sections sélectionnées dans la carte de processus d'assemblage par association, les opérateurs d'assemblage appliqués ainsi que les autres actions exécutées. La Figure 144 visualise également la démarche menée sur la carte de processus d'assemblage par association en faisant apparaître l'ordre dans lequel les sections ont été choisies.

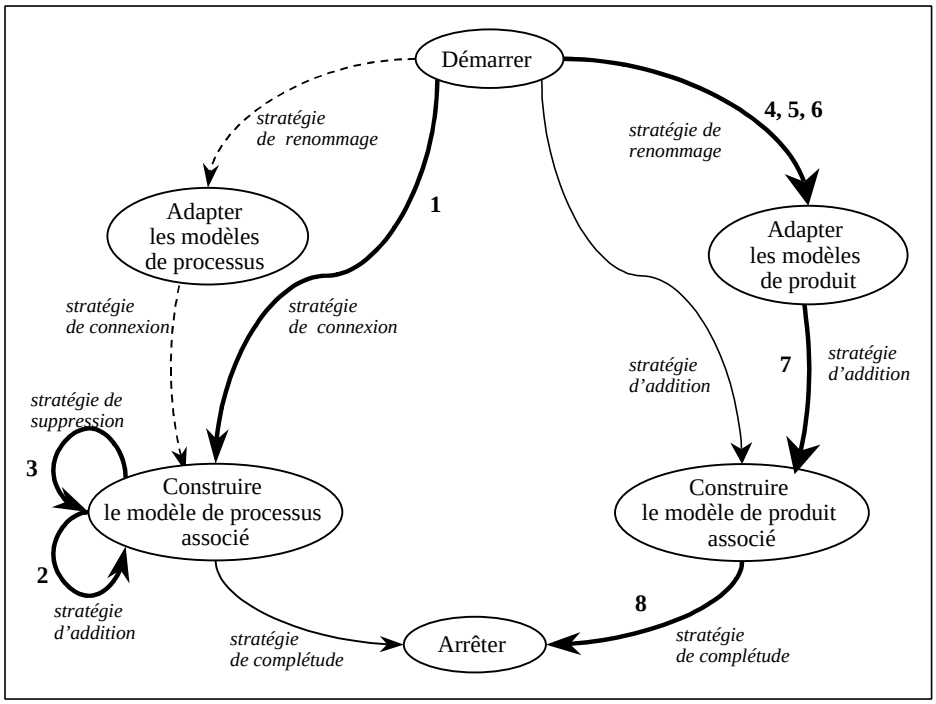


Figure 144 : Démarche utilisée pour l'assemblage des composants CREWS-*L'Ecritoire* et *Albert*

N°	Section sélectionnée	Opérateur d'assemblage	Actions
1	<Démarrer, Construire un modèle de processus associé, Stratégie de connexion>	FUSIONNER_INTENTION (CREWS-*L'Ecritoire*, Conceptualiser un scénario, Albert, Démarrer)	Identification d'une intention source dans CREWS-*L'Ecritoire* : *Conceptualiser un scénario* Fusion de cette intention avec l'intention *Démarrer* du composant *Albert*.
2	<Construire un modèle de processus associé, Construire un modèle de processus associé, Stratégie d'addition>	AJOUTER_SECTION(MPcA, <Conceptualiser un scénario, Conceptualiser un scénario, Stratégie de complétude>)	Addition d'une section <Conceptualiser un scénario, Conceptualiser un scénario, Stratégie de complétude> dans le modèle de processus dont la DRI associée est basée sur la DRI de la section <Conceptualiser un scénario, Arrêter, Stratégie de complétude>
3	<Construire un modèle de processus associé, Construire un modèle de processus associé, Stratégie de suppression>	SUPPRIMER_SECTION(MPcI, <Conceptualiser un scénario, Arrêter, Stratégie de complétude>)	Suppression de la section <Conceptualiser un scénario, Arrêter, Stratégie de complétude> Suppression de la DRI : <*({Scénario}), Arrêter avec la stratégie de complétude*>. Modification de la DSI associée à l'intention *Conceptualiser un scénario*.
4	<Démarrer, Adapter les modèles de produit, Stratégie d'unification >	RENOMMER_CONCEPT(Albert, Agent, Agent_Albert)	Changement du nom du concept *Agent* en *Agent_Albert* dans le modèle de produit d'*Albert.*
5	<Démarrer, Adapter les modèles de produit, Stratégie d'unification >	RENOMMER_CONCEPT(Albert, Etat, Etat d'agent)	Changement du nom du concept *Etat* en *Etat d'agent* dans le modèle de produit d'*Albert*.
6	<Démarrer, Adapter les modèles de produit, Stratégie d'unification >	RENOMMER_CONCEPT(Albert, Action, Action formelle)	Changement du nom du concept *Action* en *Action formelle* dans le modèle de produit d'*Albert.*
7	<Adapter les modèles de produit, Construire un modèle de produit associé, Stratégie d'addition>	AJOUTER_LIEN(MpdA, Correspond à(Agent, Agent_Albert)	Création d'un nouveau lien entre les concepts *Agent* et *Agent_Albert*.

8	<Construire un modèle de produit associé, Arrêter, Stratégie de complétude>		Vérification de la complétude du composant obtenu.

Tableau 8 : Résumé de la démarche appliquée lors de l'assemblage des composants CREWS-*L'Ecritoire* et *Albert*

4. CONCLUSION

Dans ce chapitre nous proposons une approche de sélection et d'assemblage des composants de méthode aidant l'ingénieur d'applications à satisfaire les objectifs suivants :

- construire une nouvelle méthode conforme aux exigences en cours,
- enrichir une méthode existante par une nouvelle démarche empruntée à une autre méthode et
- étendre une méthode existante avec une nouvelle fonctionnalité.

Puisque toute méthode est un composant de type agrégat (Chapitre 3) nous avons considéré une méthode entière comme un composant de méthode. Ceci nous a permis de définir un modèle de processus universel, applicable sur tous les types de composants.

Le choix de la modularité présent dans la première partie du mémoire a été conservé dans la deuxième partie. Ceci nous a permis de préserver l'uniformité de la présentation de l'approche proposée dans ce travail ainsi que de réutiliser du principe de la description modulaire des méthodes dans la présentation du modèle de processus d'assemblage. Nous avons donc formalisé celui-ci sous forme d'une carte sur laquelle nous avons appliqué le principe de décomposition en composants proposé dans le Chapitre 4. Ceci nous a permis de le présenter sous forme de composants d'assemblage qui sont d'un niveau supérieur à celui des composants de méthode (car ils se préoccupent de l'assemblage des composants de méthode afin d'obtenir des nouveaux composants de méthode). Chacun de ces composants traite un cas spécifique d'assemblage. En combinant ces composants nous pouvons réaliser tous les types d'assemblage et assembler tous les types de composants de méthode.

Dans ce chapitre, nous montrons que toute méthode formalisée sous forme de composants de méthode peut être enrichie et étendue par des nouveaux composants grâce à notre modèle de processus d'assemblage. Ceci est valable également pour le modèle de processus d'assemblage lui-même. Si nécessaire, sa structure à base des composants permet de l'étendre par des nouveaux composants d'assemblage.

Nous développons dans ce travail cinq principaux composants d'assemblage et nous illustrons leurs applications à des cas concrets.

CHAPITRE 7

Base de composants de méthodes CREWS

Ce chapitre présente un environnement permettant de faciliter la compréhension et l'utilisation des méthodes d'ingénierie de systèmes. Il se présente sous forme d'un guide électronique centré sur une base de composants de méthodes exploitant la dimension modulaire des méthodes. Cette base a été développée dans le cadre du projet de recherche européen CREWS. Par conséquent, tous les composants ainsi que l'environnement de la base sont présentés en anglais.

1. INTRODUCTION

Dans le projet CREWS nous avons développé quatre approches à base de scénarios qui sont destinées à la découverte et la validation des besoins. Ces approches ont été décrites sous forme de composants de méthodes suivant le méta-modèle proposé au Chapitre 3. L'hypothèse du projet est que chacune d'elles peut être utile dans des situations spécifiques qui ne sont pas prises en compte par des méthodes existantes et que ces approches pourraient être intégrées dans ces méthodes. Ceci a émergé un besoin d'un environnement pour consulter et sélectionner les composants de méthodes adaptés à la situation du projet en cours. Par conséquent, nous proposons de développer un guide méthodologique électronique fondé sur une base de composants de méthodes et accessible par Internet. Nous avons élargi le domaine méthodologique à tous les types de composants de méthodes.

Nous proposons de développer un environnement ayant des propriétés suivantes :

- un accès facile aux composants de méthodes stockés dans la base,

- une navigation à travers les composants,
- une mise à jour de la base de méthodes.

L'environnement proposé pour notre base de méthodes est un environnement Internet composé de deux sous environnements mis à la disposition, respectivement :

1. de l'ingénieur d'applications pour l'accès aux composants de méthodes, et
2. de l'ingénieur de méthodes pour la mise à jour de la base.

La Figure 145 illustre comment ces deux environnements sont organisés autour de notre base de méthodes afin de permettre l'accès à la base aux ingénieurs de méthodes et aux ingénieurs d'applications. L'ingénieur de méthodes enrichit la base en ajoutant de nouveaux composants tandis que l'ingénieur d'applications peut extraire, consulter et assembler les composants dont il a besoin. Le stockage, l'extraction et la consultation des composants sont les principales fonctions de la base de méthodes.

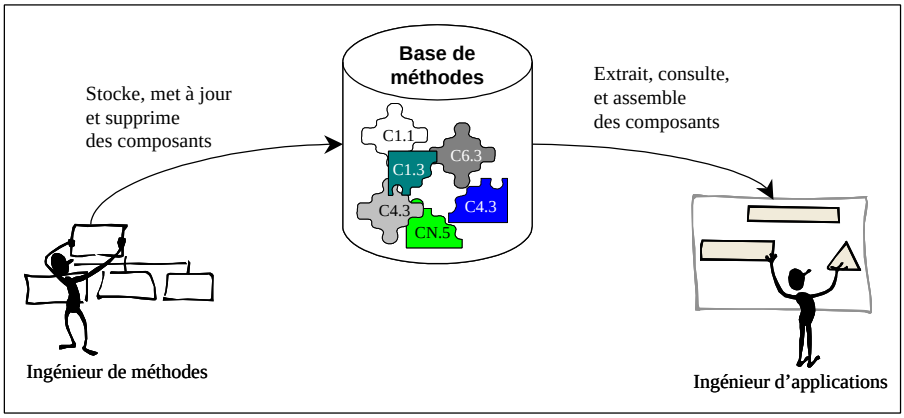


Figure 145 : Utilisation de la base de composants de méthode

Ce chapitre illustre l'implémentation de la base de composants de méthodes en SGML/HTML et son application sur la méthode CREWS-*L'Ecritoire*. La section 2 présente la structure de notre base de composants de méthodes. La section 3 présente l'architecture de cette base. Les sections 4 et 5 décrivent respectivement les deux environnements dédiés à l'ingénieur d'applications et l'ingénieur de méthodes.

2. STRUCTURE DE LA BASE DE COMPOSANTS DE METHODE

Notre base de méthodes est composée de deux parties. La première concerne la connaissance de niveau méthode, la connaissance réutilisable qui représente le contenu des composants. La seconde concerne la méta-connaissance, la connaissance nécessaire pour sélectionner et extraire les composants de la base de méthodes.

La première partie définit chaque composant de méthode par le moyen d'un ou plusieurs documents HTML (Hyper Text Markup Language) qui représentent le contenu de celui-ci : les parties de produit utilisées ainsi que sa directive décrite de manière informelle et/ou représentées graphiquement respectant le type de la directive. Des liens vers d'autres documents HTML souvent sont proposés afin de pouvoir accéder directement des sous-composants d'un composant agrégat et/ou des super-composants d'un composant, s'il fait partie d'un composant agrégat, ainsi que le glossaire des termes utilisés dans la représentation du composant.

La seconde partie représente la connaissance nécessaire à l'extraction des composants de la base, c'est-à-dire le descripteur et la signature de chaque composant par le moyen d'un document SGML (Standard Generalized Markup Language) [Lemaitre 95]. Nous utilisons le langage SGMLQL pour construire des requêtes permettant d'extraire les composants de la base de méthodes.

La partie SGML est identique pour chaque composant. La structure HTML peut être adaptée en fonction du contenu de chaque composant.

2.1 Représentation des composants de méthode en HTML

La représentation HTML d'un composant comporte non seulement son corps mais aussi sa signature pour mieux comprendre le contexte d'application du composant ainsi que son descripteur qui définie le contexte de sa réutilisation.

Chaque composant a un *nom* qui correspond à l'intention du composant décrite de manière informelle et est structuré en un ensemble de sections :

- l'*objectif* qui explique à quoi sert le composant,
- la *situation* qui désigne les parties de produit nécessaires à l'application du composant ainsi que les pré-conditions associées,
- l'*intention* qui spécifie la partie intention du composant,
- la *représentation graphique* qui illustre la directive du composant par une figure correspondant au type de la directive (plan, choix, carte, ensemble d'actions).
- la *description* qui donne une explication concernant la représentation graphique de la directive ou le contenu de la directive informelle,
- le *type* qui précise si le composant est atomique ou agrégat,
- la section *agrégats* qui référence, s'ils existent, les composants auxquels appartient le composant considéré,
- la section *composants* qui référence les sous-composants, s'ils existent, contenu dans le composant considéré,

- le *contexte de réutilisation* qui précise dans quel domaine d'application le composant peut être utilisé ainsi que l'activité de conception qui peut être réalisée par ce composant,

- l'*origine* qui comporte le nom de la méthode d'origine du composant ainsi que les références bibliographiques des documents qui décrivent cette méthode,

- le *support d'apprentissage* qui comporte des exemples d'application du composant,

- l'*expérience* qui comporte les remarques et les suggestions des ingénieurs d'applications qui ont déjà utilisé ce composant.

Le document HTML du chaque composant a une structure suivante : au sommet de la page nous trouvons des liens vers les différentes sections mentionnés ci-dessus. En cliquant sur ces liens nous pouvons visualiser ces sections. Cliquer sur le lien "Intention", par exemple, met en avance la section du document décrivant l'intention du composant. Puisque l'intention du composant a une structure prédéfinie (verbe, cible, manière, etc.) (voir le Chapitre 3), chaque élément de l'intention peut être défini comme un lien vers un autre document HTML, celui qui comporte le glossaire de la base de composants de méthodes. Ce glossaire propose les définitions de tous les verbes et de toutes les cibles utilisés dans les intentions des composants de la base ainsi que des synonymes possibles et des exemples d'utilisation. La Figure 146 illustre un composant de méthode représenté par une page HTML avec des liens vers le glossaire et la description du produit correspondant.

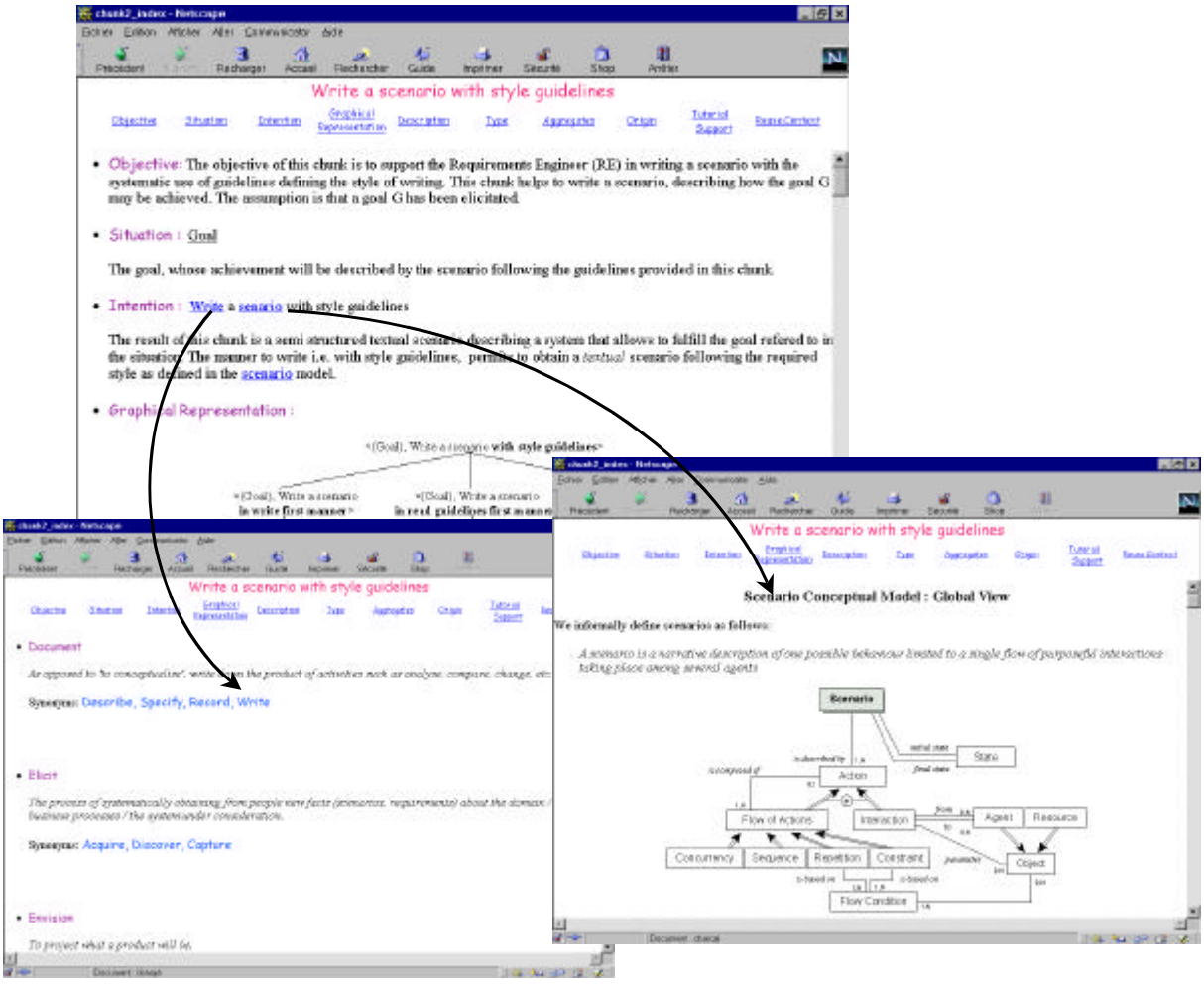


Figure 146 : Exemple de représentation d'un composant de méthode

Si le composant est de type agrégat, nous pouvons accéder directement les documents HTML de ses sous-composants en cliquant sur les différentes parties de sa représentation graphique ou sur les liens listés dans la section *"Composants"*. De la même manière, si le composant fait partie d'un ou plusieurs d'autres composants, nous pouvons visualiser les documents HTML de ses super-composants en cliquant sur les liens listés dans la section "Agrégats". La Figure 147 illustre la représentation HTML de trois composants de méthodes. Le composant représenté au sommet de la figure est un agrégat. Les deux autres composants sont ses sous-composants. Puisque la directive du composant du départ est représentée par une carte, toutes les sections de cette carte sont également des composants de niveau de granularité plus fin. Ainsi nous pouvons accéder ses sous-composants en cliquant sur les différentes parties de la représentation graphique du composant, c'est-à-dire sur les sections de sa carte. Ceci est également possible en passant par la section *"Composants"* dans la page HTML du composant agrégat où sont lister tous ses sous-composants en cliquant sur le nom du sous-composant que l'on veut visualiser.

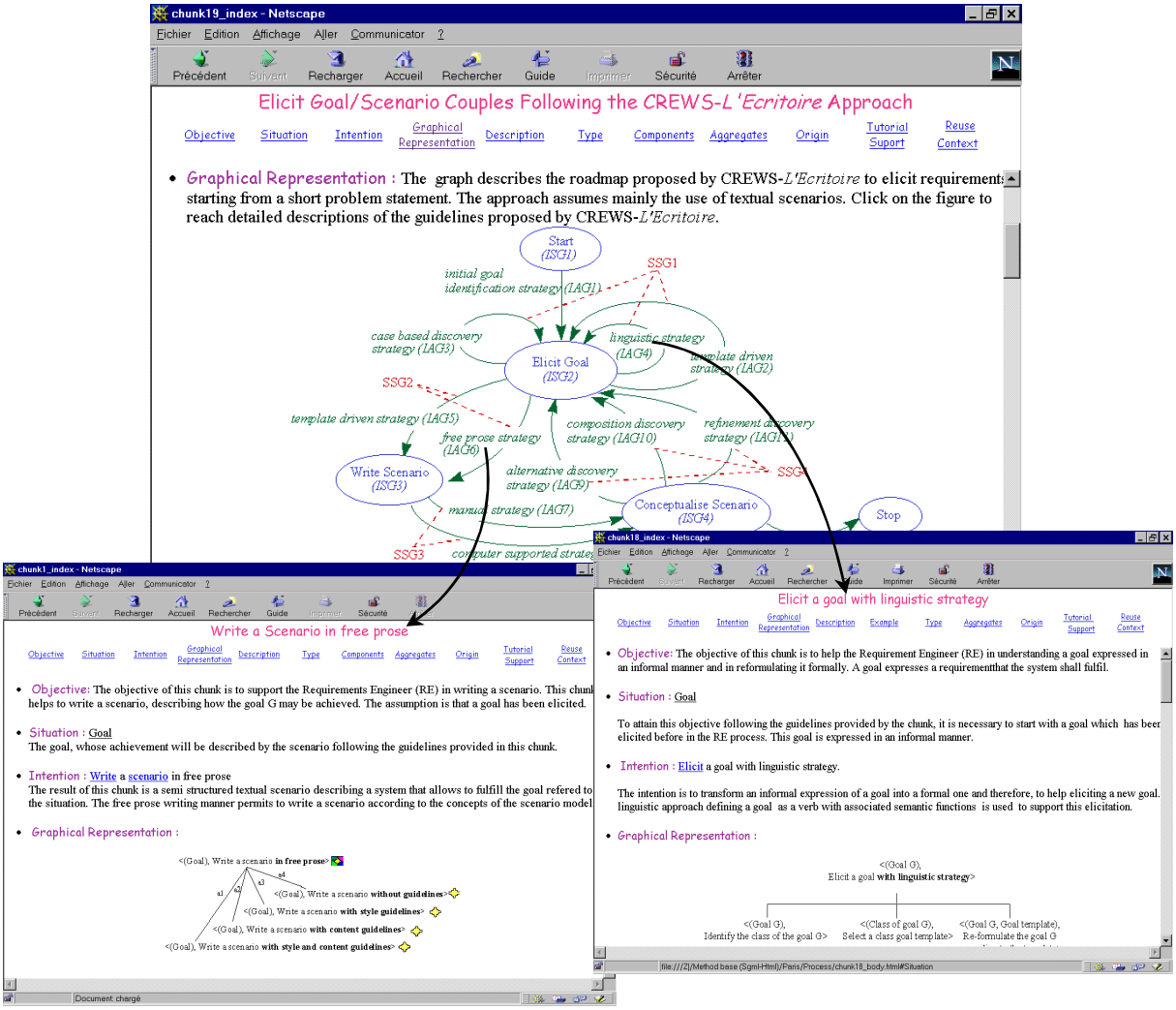


Figure 147 : Exemple de navigation entre les composants de méthodes

2.2 Représentation SGML de la base de méthodes

La deuxième partie de la base de composants de méthodes est représentée sous forme d'un document SGML dans lequel sont définies les informations nécessaires à l'extraction des composants. Le choix de ce langage a été motivé par sa simplicité et sa facilité d'utilisation et d'évolution. SGML [Golgfarb 90] est un langage standard pour décrire des documents structurés en utilisant un ensemble des étiquettes définies dans une grammaire. Les documents SGML sont représentés sous forme d'arbres et sont associés au langage d'interrogation SGMLQL.

La structure SGML de la base de méthodes CREWS est représentée par un arbre à la Figure 148. La racine de cet arbre est un élément CREWS-BASE (BASE-CREWS) qui est représentée par une collection des CHUNK* (composants) (une étoile à côté du nom d'une étiquette représente le fait que l'étiquette puisse être répétée plusieurs fois). L'élément CHUNK (composant) est caractérisé par un attribut *kind* (type) et des étiquettes : INDEX, NAME (nom), PRODUCT-MODEL (modèle de produit), DESCRIPTOR (descripteur), HTML-BODY (corps HTML), COMPONENT (sous-composant), AGGREGATE (agrégat) et ORIGIN (origine). L'INDEX est un identificateur du composant. L'étiquette NAME contient le nom du composant qui représente son intention d'une manière informelle. Le PRODUCT-MODEL contient le nom du modèle de produit correspondant. L'étiquette DESCRIPTOR est composée de deux étiquettes DESCRIPTOR-SITUATION (situation de descripteur) et DESCRIPTOR-INTENTION (intention de descripteur).

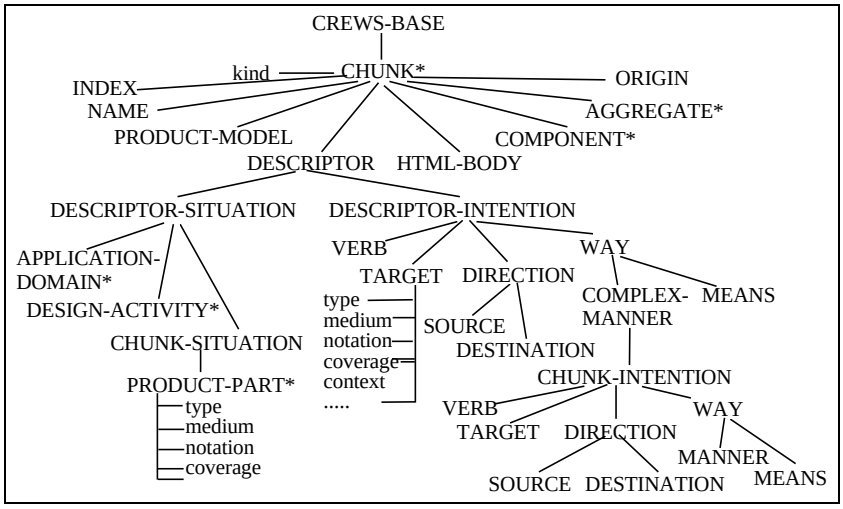


Figure 148: Structure de la partie SGML de la base de méthodes

Comme le montre la Figure 148, la situation de descripteur DESCRIPTOR-SITUATION a trois parties : le domaine d'application APPLICATION-DOMAIN, l'activité de conception DESIGN-ACTIVITY et aussi elle intègre la situation du composant correspondant CHUNK-SITUATION. Chaque composant peut être appliqué dans un ou plusieurs domaines d'application dans la réalisation d'une ou plusieurs activités de conception. L'étiquette CHUNK-SITUATION détermine quelles sont les parties de produit PRODUCT-PART nécessaires pour appliquer le composant. Les parties de produit sont classées en deux types : "Scénario" et "Non-scénario" à l'aide d'un attribut "type". Les parties de produit de type "scénario" sont caractérisées par un ensemble d'attributs. Ces attributs sont définit dans un cadre de référence de classification des approches par scénarios [Rolland 98a]. Ces

attributs permettent de définir quel est le moyen d'expression des scénarios (texte, tableau, graphique, image etc.), la notation (informelle, formelle, semi-formelle), la couverture (fonctionnelle, non fonctionnelle, intentionnelle) etc.

L'intention du descripteur DESCRIPTOR-INTENTION est décomposée en un verbe VERB, une cible TARGET, une direction DIRECTION et un chemin WAY selon la structure prédéfinie de but [Prat 97]. Les étiquettes VERB et TARGET doivent être complétées obligatoirement tandis que l'étiquette DIRECTION est optionnelle. Le paramètre WAY est composé de deux sous - paramètres : la manière complexe COMPLEX-MANNER et le moyen MEANS. L'étiquette COMPLEX-MANNER doit être complétée obligatoirement car elle exprime l'intention du composant d'une manière récursive. Elle a aussi la structure prédéfinie de but, c'est-à-dire, le verbe VERB, la cible TARGET et les autres paramètres. Si par exemple l'intention du composant est "*Ecrire* $_{verb}$ *(un scénario)* $_{Cib}$ *(en langage naturel)* $_{Man}$" l'intention du descripteur de ce composant est "*Documenter* $_{Verb}$ *(un besoin fonctionnel du système)*$_{Cib}$ *(en écrivant (un scénario)* $_{cib}$ *(en langage naturel)* $_{Man}$ *)* $_{Man}$ ".

La Figure 149 présente la description SGML d'un composant de la base CREWS.

```
<CHUNK kind="aggregate">
<INDEX>chunk1</INDEX>
<NAME>Write a scenario in free prose</NAME>
<PRODUCT-MODEL>CREWS-L'Ecritoire</PRODUCT-MODEL>
<DESCRIPTOR><DESCRIPTOR-SITUATION>
<APPLICATION-DOMAIN>Information Systems</APPLICATION-DOMAIN>
<APPLICATION-DOMAIN>Business Processes</APPLICATION-DOMAIN>
<APPLICATION-DOMAIN>Socio-Technical Systems</APPLICATION-DOMAIN>
<APPLICATION-DOMAIN>Human Computer Interfaces</APPLICATION-DOMAIN>
<DESIGN-ACTIVITY>Requirements Elicitation</DESIGN-ACTIVITY>
<DESIGN-ACTIVITY>Analysis</DESIGN-ACTIVITY>
<DESIGN-ACTIVITY>Documentation</DESIGN-ACTIVITY>
<CHUNK-SITUATION>
<PRODUCT-PART type="Non scenario based">Goal</PRODUCT-PART>
</CHUNK-SITUATION></DESCRIPTOR-SITUATION>
<DESCRIPTOR-INTENTION>
<VERB>Document</VERB><TARGET>System requirements</TARGET>
<COMPLEX-MANNER><VERB>Write</VERB><TARGET>Scenario</TARGET>
<MANNER>Free prose</MANNER></COMPLEX-MANNER></DESCRIPTOR-INTENTION>
<ORIGIN>CREWS-L'Ecritoire</ORIGIN></DESCRIPTOR></CHUNK>
```

Figure 149 : Exemple de la description SGML du composant *<(Goal), Write a scenario in free prose>*

Il est également possible d'utiliser le langage SGML pour décrire les contenus des composants et de générer dynamiquement des pages HTML présentant les composants à l'aide de langage de requêtes SGMLQL. Mais pour le moment, le SGMLQL n'est pas suffisamment développé pour que l'on puisse obtenir des pages HTML bien structurées et aussi conviviales que ce qui est proposé actuellement.

Surtout que le monde d'Internet nous oblige à avoir une présentation des composants bien soignée, attractive et facile à manipuler. Par souci de représentation, nous avons décidé de prédéfinir les pages HTML des composants et de ne stocker dans la partie SGML que l'information nécessaire à l'extraction des composants.

2.2.1 SGMLQL

A côté de la possibilité de sélectionner des composants de méthodes à partir de la liste de composants, l'environnement d'interrogation permet d'interroger la base de méthodes à l'aide d'un langage d'interrogation nommé SGMLQL. Le langage SGMLQL [Le Maitre 95] est un langage de requêtes qui permet de questionner des documents SGML. SGMLQL sert à extraire des fragments des documents et de les réutiliser pour construire d'autres documents SGML ou HTML. IL permet également de consulter les documents distribués dans les sites distants. Nous l'utilisons pour questionner notre base de méthodes, la partie décrite en SGML.

Ce langage de requêtes proche de SQL utilise les informations liées à la structure du document pour donner le résultat de son interrogation. Comme dans les langages de type SQL, on y trouve des opérateurs tels que Select from Where, Remove, Replace… etc. La liste exhaustive des opérateurs proposés dans SGMLQL se trouve dans le manuel de référence disponible à l'adresse http ://www.lpl.univ-aix.fr/~harie. Les requêtes SGMLQL permettent d'extraire des composants de la base de méthode. La Figure 150 donne un exemple de requête.

```
global $myfile = file"MethodBase.sgml";
global $chunks=select "<LI><A HREF=".text($hb).">". text($n->NAME)."<A></LI>"
from    $c in every CHUNK within $myfile,
        $d in every DESCRIPTOR-INTENTION within $c,
        $v in first VERB within $d,
        $t in first TARGET within $d,
        $hb in HTML-BODY within $c
where text($v) match "Discover" and  text($t) match  "System functional requirements";
```

Figure 150: Exemple de requête SGMLQL

La requête représentée à la Figure 150 sélectionne les composants qui aident à la découverte des besoins fonctionnels d'un système d'information. La requête est basée sur le verbe de l'intention de descripteur dont la valeur doit être "Discover" (découvrir) et la cible dont la valeur est "Information System functional requirements" (besoins fonctionnels d'un système d'information). Le résultat de la requête est un document HTML comportant une liste des noms des composants avec des liens aux documents HTML correspondants.

3. ARCHITECTURE ET FONCTIONS

L'architecture de la base de méthodes est basée sur un environnement client-serveur utilisant le Web. L'outil est composé de deux parties : le *navigateur de composants* qui fonctionne sur le client et le *moteur de recherche et de mise à jour de composant* qui fonctionne sur le serveur. La Figure 151 montre l'architecture du prototype.

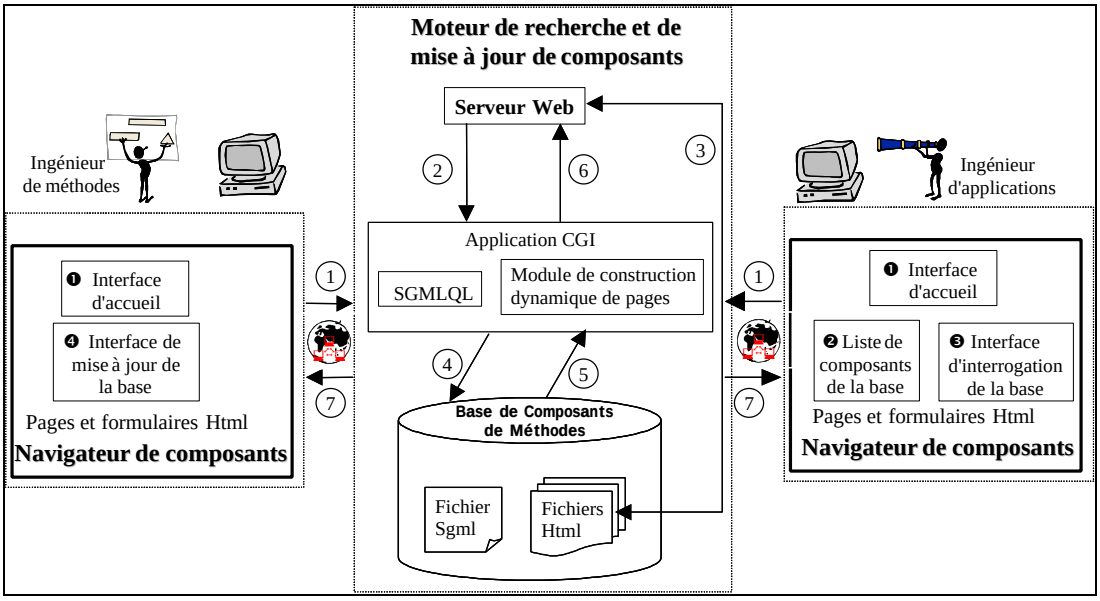


Figure 151: Architecture de l'environnement

Le *navigateur de composants* est l'interface fournie aux ingénieurs, il comprend un ensemble de pages et de formulaires HTML permettant d'accéder à la base de composants (flèche ① à la Figure 151) à travers Internet, Intranet ou localement. Les pages et les formulaires HTML peuvent être visualisés par n'importe quel navigateur Internet (Internet Explorer, Netscape navigateur, Mosaic, etc.). Tous les ingénieurs entrent dans la base de composants de méthode par l'interface d'accueil ❶. Les ingénieurs de méthodes utilisent l'interface de mise à jour de la base pour enrichir la base ❹, tandis que les ingénieurs d'application utilisent l'interface d'interrogation ❷ et la liste des composants ❸ pour consulter la base.

Le moteur de recherche et de mise à jour de composants est composé d'un serveur Web, de deux applications CGI et la base de composants de méthode. Le serveur Web est responsable de tous les échanges de communication entre le client et les applications CGI (flèche ① et ② de la Figure 151) qui s'exécutent sur le serveur. Il reçoit les demandes du client et envoie (flèche ①et ⑦ de la Figure 151) la page résultat qui est, soit générée à la volée à partir de la base de composants, soit récupérée dans la base de composants. La récupération de page HTML (flèche ③ de la Figure 151) consiste à rechercher les fichiers HTML correspondant à la demande de l'ingénieur. Par contre, la génération à la volée, consiste à exécuter un des deux CGI (flèche ② de la Figure 151). Les applications CGI font appel au module de requête SGMLQL qui interroge la base (flèche ④ de la Figure 151) et extrait les informations demandées. Ces informations sont transmises au CGI et plus particulièrement au module de construction dynamique de pages (flèche ⑤ de la Figure 151) qui met en forme ces informations sous forme de page HTML et les renvoie au serveur Web (flèche ⑦ de la Figure 151). Cette page est

alors renvoyée aux ingénieurs via le navigateur de composants, à partir duquel ils peuvent rechercher, extraire et consulter de nouveaux composants.

4. ENVIRONNEMENT DE L'INGENIEUR D'APPLICATIONS

L'utilisation de la base nécessite une interface d'interrogation permettant de rechercher des composants. Il est illustré à la Figure 152. Comme le montre cette figure, l'ingénieur d'applications peut soit sélectionner un composant à partir d'une liste de composants, soit construire des requêtes sur la base.

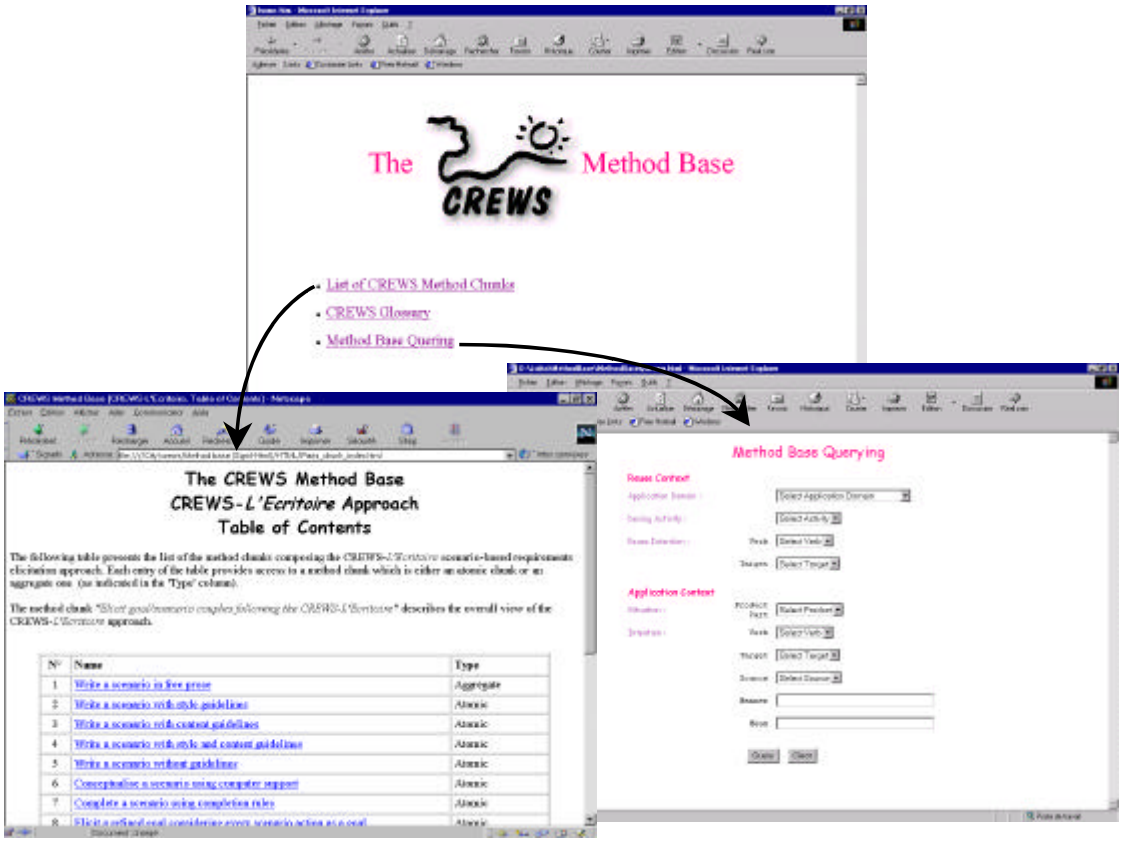


Figure 152: Les interfaces de consultation de la base

La liste des composants de méthode comprend un ensemble de composants de la méthode CREWS-*L'Ecritoire*. Cette liste permet à l'ingénieur de sélectionner le composant dont il a besoin selon l'intention et le type de composant (atomique ou agrégat). La Figure 153 montre l'interface donnant une partie de la liste des composants.

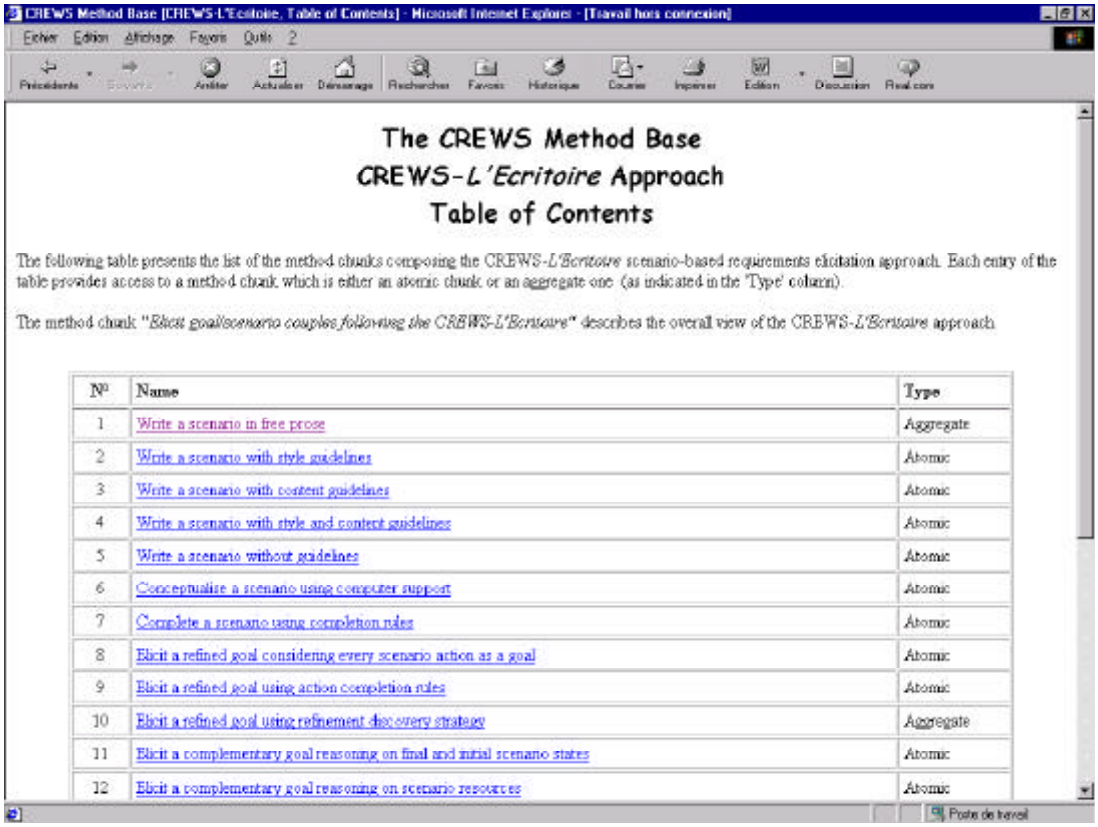


Figure 153: Interface donnant la liste des composants de la base

4.1 Formulaire d'interrogation

Dans le cas de l'interrogation, les requêtes SGMLQL ne sont pas écrites par l'ingénieur d'application mais générées automatiquement en utilisant le formulaire d'interrogation. Ce dernier permet d'accéder au contenu de la base de méthodes. Il est illustré à la Figure 154.

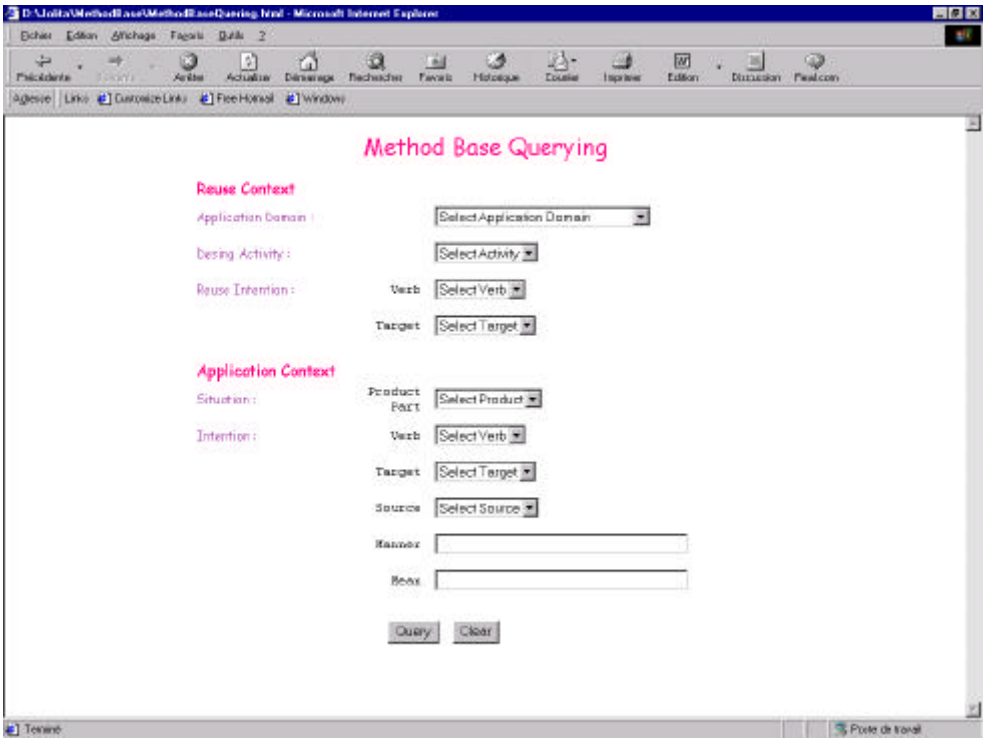


Figure 154: Interface d'interrogation de la base

Comme le montre cette figure, l'ingénieur d'applications peut interroger la base de méthodes en proposant des valeurs à des paramètres du contexte de réutilisation ou/et du contexte d'application. Le contexte de réutilisation comporte les différentes facettes définies dans le descripteur tandis que le contexte d'application est basé sur les signatures des composants. L'ingénieur d'applications peut interroger la base suivant un seul des paramètres de l'interface ou par combinaison de plusieurs paramètres. Le paramètre *Intention de réutilisation* est décomposé en sous-paramètres *Verbe, Cible.* Le paramètre Intention dans le contexte d'application est décomposé en sous-paramètres *Verbe, Cible, Source, Manière, Moyen.* Pour la situation, l'ingénieur d'applications peut définir un élément de la *partie de produit* parmi une liste d'éléments correspondant aux différentes parties de produit des situations des composants. Pour chacun des paramètres, l'interface d'interrogation propose un champ associé à ce paramètre et une liste de valeurs prédéfinies parmi lesquelles l'ingénieur d'applications peut choisir. Les requêtes SGMLlQL sont exécutées sur la base et le résultat transmis sous forme d'une page HTML à l'ingénieur d'applications.

Supposons, que l'ingénieur d'application décide de sélectionner les composants de méthodes ayant comme intention *"Découvrir un but"* dans le domaine d'application *"Systèmes d'information"*. Il remplit les paramètres *Domaine d'application = Systèmes d'information (Application Domain = Information Systems)* dans la partie *Contexte de réutilisation* du formulaire et les paramètres *Verbe = Découvrir* et *Cible = But* (*Verb = Elicit, Target = Goal*) dans la partie *Contexte d'application* du formulaire. La requête générée par le CGI à partir du formulaire est illustrée à la Figure 155.

```
global $myfile = file"MethodBase.sgml";
global $chunks=select "<LI><A HREF=".text($hb).">". text($n->NAME)."<A></LI>"
from    $c in every CHUNK within $myfile,
```

```
        $ds in everv DESCRIPTOR-SITUATION within $c.
        $ad in every APPLICATION-DOMAIN within $ds,
        $di in every DESCRIPTOR-INTENTION within $c,
        $ci in every CHIUNK-INTENTION within $di,
        $v in first VERB within $ci,
        $t in first TARGET within $ci,
        $hb in HTML-BODY within $c
where text($ad) match "Information Systems" text($v) match "Elicit" and  text($t) match  "Goal";
```

Figure 155 : Requête générée à partir du formulaire

Le résultat de la requête est une liste des noms de composants illustrée à la Figure 156.

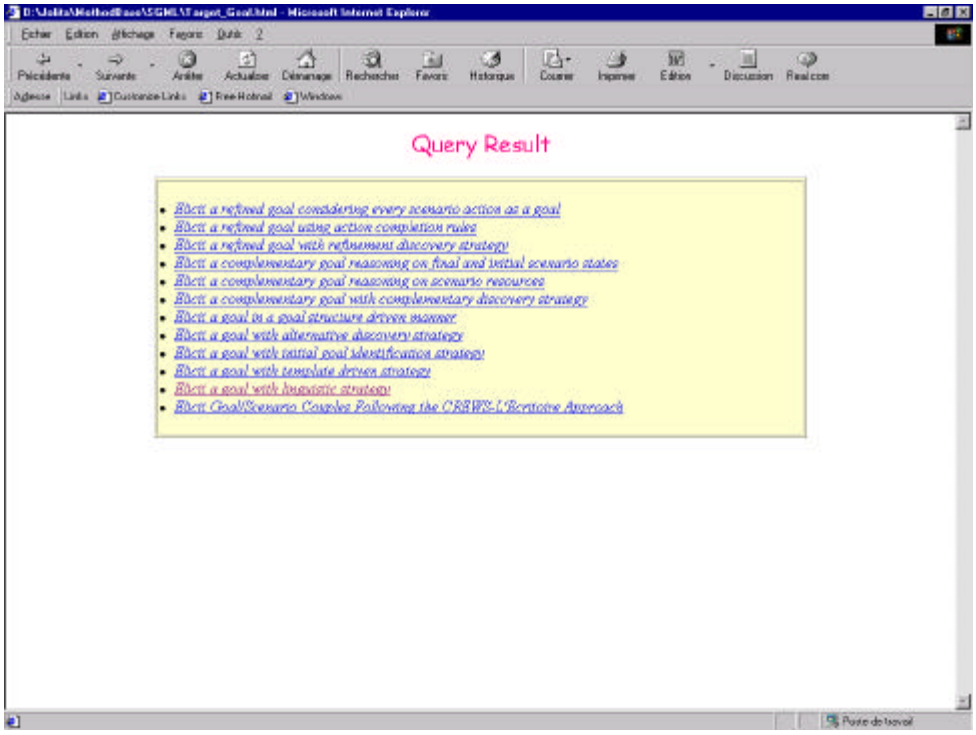


Figure 156: Exemple de résultat de l'interface d'interrogation de la base

Cet exemple montre qu'il y a douze composants correspondant à la requête. A partir de ce résultat l'ingénieur d'applications peut sélectionner un composant et accéder son corps en cliquant sur son nom dans la liste.

4.2 Navigation

L'ingénieur d'applications peut naviguer à travers les composants soit à partir de la liste des composants soit à partir du résultat des requêtes. La Figure 157 illustre le premier cas et la Figure 158 le second.

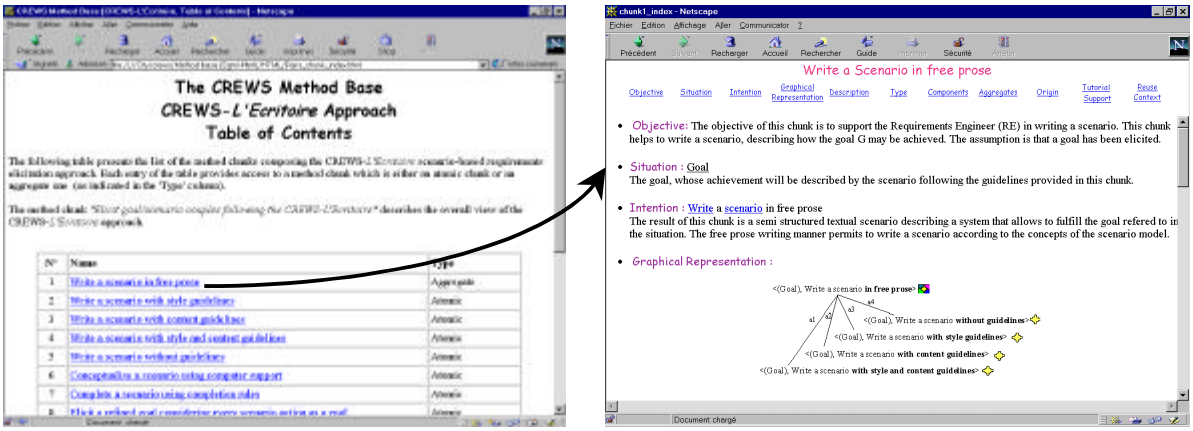


Figure 157: Exemple de navigation à partir de la liste des composants

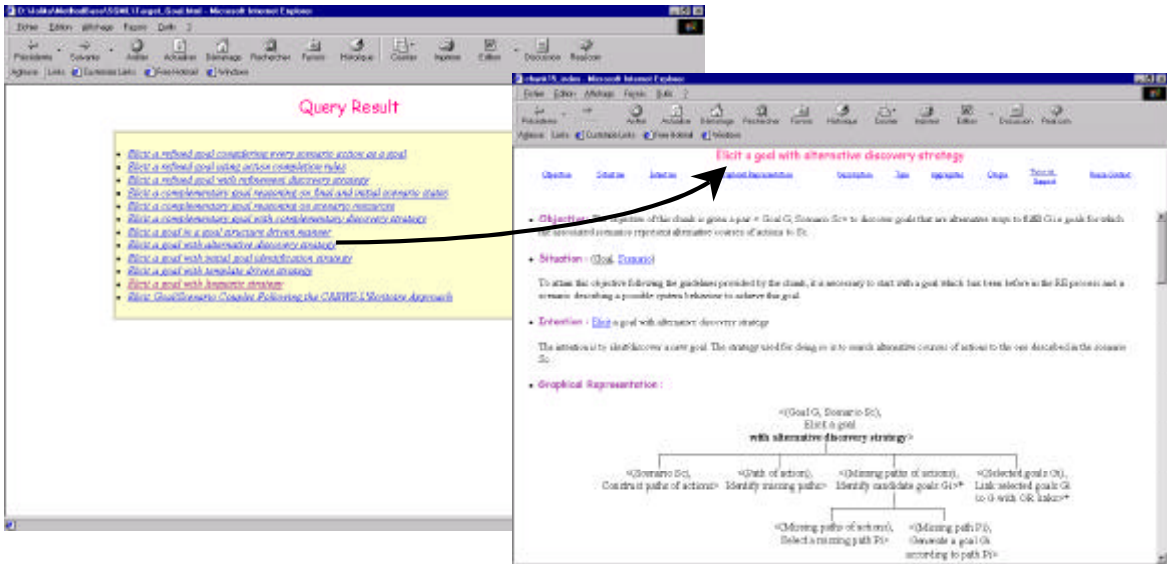


Figure 158: Exemple de navigation à partir de résultat de l'interface d'interrogation de la base

De plus, les différents liens contenus dans les pages HTML permettent de naviguer de composant en composant. Par exemple, la partie représentation graphique de la page HTML du composant permet de naviguer dans les sous-composants du composant considéré. Puis de revenir sur le composant courant etc. La Figure 159 présente un exemple de navigation à travers les composants.

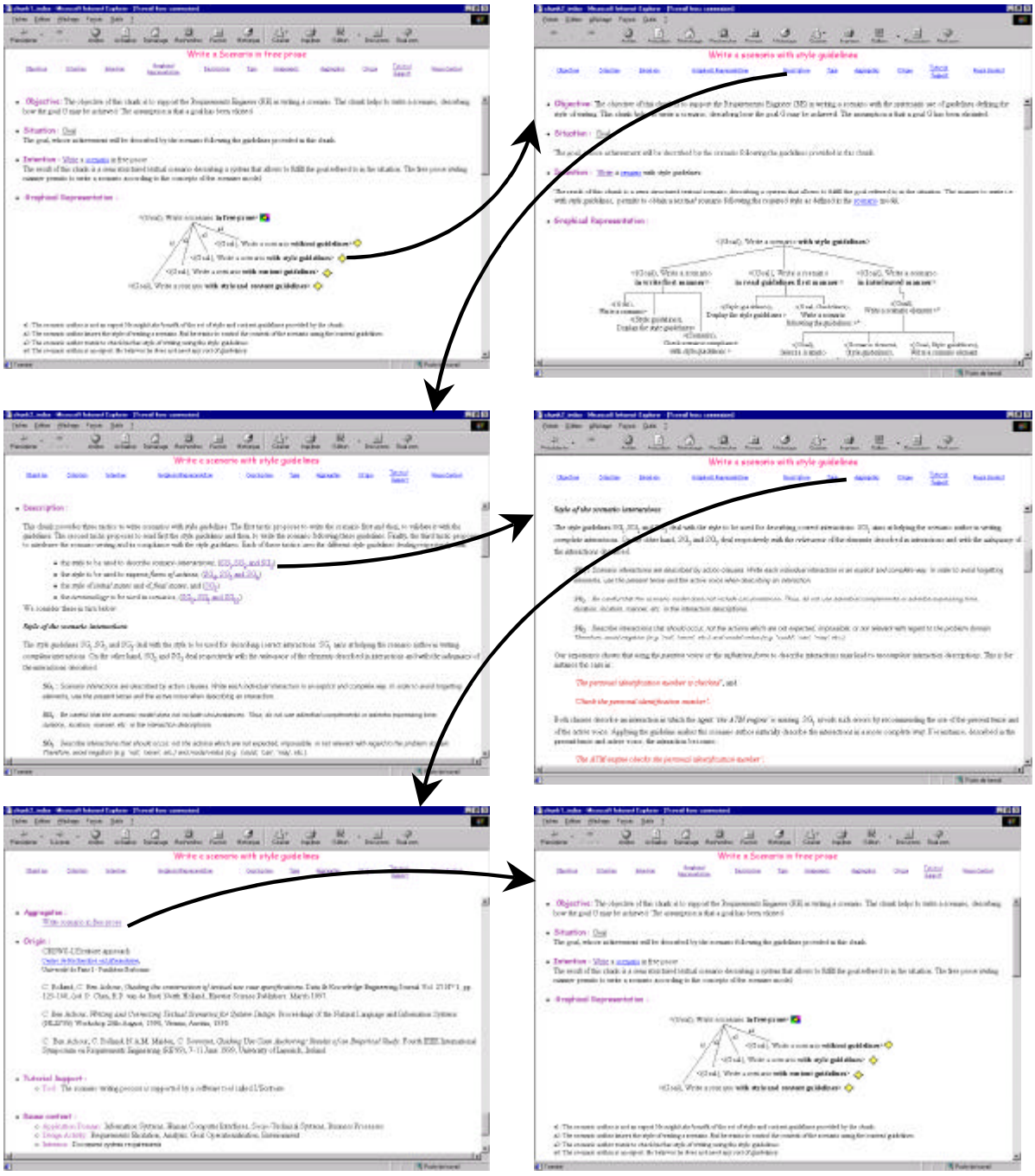


Figure 159 : Exemple de navigation dans la base de composants

La Figure 159 illustre une navigation à partir du composant dont la signature est *<(Problem statement), Write scenario in free prose>*. C'est un composant agrégat dont la directive est de type choix comportant quatre alternatives. Chacune des alternatives est également un composant. En cliquant sur la représentation graphique du composant initial on visualise un de ses sous-composant dont la signature est *<(Problem statement), Write scenario with style guidelines>*. En cliquant sur le lien *Description* de ce dernier on accède à la description textuelle du composant comportant les directives de style d'écriture des scénarios. On peut également naviguer dans cette description pour visualiser toutes ses parties. En cliquant sur le lien *Agrégats* du composant on peut revenir sur le composant du départ. Cet exemple montre comment l'ingénieur peut naviguer de composant en

composant. La navigation s'effectue à l'aide des liens hypertexte contenus dans la description HTML des composants. La facilité de navigation offerte par HTML est ce qui a motivé sa sélection.

5. ENVIRONNEMENT DE L'INGENIEUR DE METHODES

Afin de faciliter le travail de l'ingénieur de méthodes, nous avons prévu que l'environnement de notre base de méthodes devrait comporter une interface destinée à la mise à jour cette dernière. Cette interface est actuellement en cours de construction. Elle va être accessible uniquement aux ingénieurs de méthodes et pas au utilisateurs de la base.

L'interface d'ingénieur de méthodes permettra de mettre à jour le fichier SGML de la base en y ajoutant des nouveaux composants. Elle ne permettra pas de créer le corps HTML du composant mais seulement sa partie SGML. L'ingénieur de méthodes devra à l'aide de la structure prédéfinie décrire le corps du composant dans un ou plusieurs fichiers HTML.

6. CONCLUSION

La base de composants de méthodes présentée dans ce chapitre permet de mettre en œuvre l'approche modulaire présentée dans ce mémoire. Cette base guide l'ingénieur d'applications et l'ingénieur de méthodes pour rechercher, extraire et consulter les différents composants de méthodes. La description modulaire sous forme de composants permet à l'ingénieur d'applications d'extraire et d'utiliser le composant le plus adapté aux besoins spécifiques d'une situation à laquelle il est confronté.

L'implémentation de la base est centrée sur une architecture client-serveur Web utilisant le langage SGML et HTML. Le langage SGML et le langage de requêtes SGMLQL qui lui est associé permettent une description et une interrogation facile des composants de méthodes. La description HTML des composants permet grâce aux liens hypertexte de visualiser les composants dans n'importe quel navigateur Web, et ainsi de naviguer de manière simple de composant en composant.

CHAPITRE 8

Conclusion

Dans ce mémoire, nous avons abordé la construction des méthodes d'ingénierie de systèmes à base de composants. Ce sujet est la préoccupation centrale du domaine de l'*ingénierie des méthodes situationnelles*. Notre apport dans ce domaine est la formalisation des méthodes existantes sous forme de composants réutilisables et la réutilisation de ces composants dans la construction de nouvelles méthodes.

1. CONTRIBUTIONS

La solution que nous proposons dans ce mémoire est constituée de deux parties. Dans la première nous proposons une approche de re-ingénierie des méthodes existantes sous forme de composants de méthode. Dans la seconde nous proposons une technique d'assemblage des composants définis par l'application de l'approche proposée dans la première partie.

Notre approche de ré-ingénierie des méthodes est composée d'un *méta-modèle de méthodes* et d'un *modèle de processus de ré-ingénierie*.

Le *méta-modèle de méthodes* assure la description de tous les types de méthodes d'ingénierie des systèmes de manière modulaire. Il comporte un ensemble de concepts qui permettent la description du produit proposé par la méthode et de la démarche que la méthode offre pour construire ce produit. La modularité de la description est acquise grâce aux concepts de *directive* et de *partie de produit*. Le modèle de processus d'une méthode est vu comme un ensemble de directives de différents niveaux de granularité. L'association d'une directive aux parties de produit nécessaires à sa réalisation représente un composant de méthode. Le concept de *descripteur* est utilisé pour définir le contexte de réutilisation du composant et pour faciliter l'extraction des composants de la base de méthodes.

Le *modèle de processus de ré-ingénierie de méthodes* offre le guidage dans la reconstruction de méthodes conformément au méta-modèle de méthodes proposé.

La technique d'assemblage de composants de méthodes que nous avons proposé dans la deuxième partie de la thèse est fondée sur trois éléments : les *opérateurs d'assemblage*, les *mesures de similarité* et les *règles de validation de la qualité* de l'assemblage. Tous ces éléments sont utilisés par le *modèle de processus d'assemblage*.

Le processus d'assemblage est basé sur l'application de deux types *d'opérateurs d'assemblages* : les opérateurs permettant d'assembler les parties de produit des composants et les opérateurs utilisés pour assembler leurs directives.

Les *mesures de similarité* contribuent à l'identification des parties communes dans différents composants de méthode et à l'adaptation des parties de produit et des directives des composants pour assurer la conformité de leur assemblage.

Les *règles de qualité* servent à valider la cohérence de la démarche choisie par l'ingénieur d'applications lors de l'assemblage des composants ainsi que la complétude du résultat obtenu.

Le *modèle de processus d'assemblage* offre le guidage dans la sélection des composants de méthode et dans leur assemblage. Il permet de satisfaire trois objectifs : la construction d'une nouvelle méthode à partir de différents composants de méthodes, conformément aux exigences du projet en cours, la complétude de la démarche d'une méthode existante par celle d'un autre composant de méthode et l'extension d'une méthode existante par une nouvelle fonctionnalité empruntée à un autre composant de méthode. De plus, il inclut deux modes d'assemblage : *l'intégration* et *l'association*. Le premier prend en compte l'assemblage de composants disjoints tandis que le second permet de considérer le cas de composants qui se recouvrent partiellement.

La présentation du modèle de processus d'assemblage sous forme des composants d'assemblage offre la possibilité de l'étendre par des nouveaux composants qui pourront être définis ultérieurement.

2. PERSPECTIVES

Le travail présenté dans ce mémoire peut être poursuivi dans plusieurs directions.

- Evolution du modèle processus de de ré-ingénierie de méthodes.

  D'un coté, de nouvelles stratégies concernant la reconstruction d'une méthode sous forme d'une carte et la définition de composants de méthode à partir de cette carte peuvent être ajoutées à ce modèle afin de l'enrichir.

  D'un autre coté, les stratégies existantes peuvent être développées jusqu'à la description des actions atomiques à exécuter afin d'améliorer le guidage qu'elles proposent. Ceci serait un pas vers l'automatisation du processus de définition des composants de méthodes.

Ce modèle à été appliqué à la construction la méthode CREWS-*L'Ecritoire* à sa présentation sous forme de composants. Nous l'avons également validé en l'appliquant sur la méthode OOSE. Il serait intéressant de le valider sur d'autres types de méthodes. Par exemple, les méthodes proposant des démarches coopératives qui font intervenir plusieurs décideurs ou les méthodes d'application temps réel qui nécessitent la spécification de contraintes de synchronisation strictes. La modélisation de telles méthodes pourrait faire apparaître la nécessité d'étendre le modèle de ré-ingénierie de méthodes, afin de tenir compte des particularités de ces démarches.

- Evolution du modèle de processus d'assemblage.

  De nouveaux cas d'assemblage peuvent être définis sous forme de composants et introduits dans le modèle de processus d'assemblage en tant que nouvelles stratégies.

- Complétude de la base de méthodes.

Actuellement, notre base de méthodes comporte des composants issus des méthodes CREWS-*L'Ecritoire* et OOSE. Pour rendre fonctionnelle notre base de méthodes et pouvoir appliquer notre technique d'assemblage il est nécessaire de la compléter par des composants de plusieurs méthodes.

- Le résultat de cette thèse peut être poursuivi sur le plan pratique, par le développement d'un environnement informatisé supportant :

  ✓ la reconstruction des méthodes sous forme de composants, par exploitation du modèle de processus de ré-ingénierie de méthodes et le méta-modèle de méthodes modulaires ;

  ✓ l'assemblage de composants de méthodes, par exploitation du modèle de processus d'assemblage.

Le développement de tels outils nous semble nécessaire pour réellement faciliter le travail de l'ingénieur d'applications et de l'ingénieur de méthodes.

Un environnement logiciel appelé MAP est en cours de construction au CRI[7] à l'université Paris 1 - Sorbonne. Cet environnement propose des fonctionnalités de base concernant la gestion des processus décrits sous forme de cartes comme par exemple, une interface de saisie et la validation de cartes, un moteur d'exécution de processus modélisés par des cartes, etc. La ré-ingénierie des méthodes et l'assemblage de composants pourraient être définis en tant que fonctionnalités supplémentaires de cet environnement.

[7] Centre de Recherche en Informatique

ANNEXE

The CREWS Glossary

1. PRODUCT

Term	Definition	Synonyms
Actor	An actor is a kind or a category of users. He defines the role that user can play. There are two kinds of actors: the primary actors who use directly the system, each one do one or more principles tacks of the system. The secondary actors permit to the primary ones to use the system.	
Animation	An animation is the creation of a finite set of finite sequences describing normative or non-normative behaviours of the composite system. While scenarios focus on the interactions taking place between the system and its environment, the result of the animation considers the possible behaviours of the whole composite system and helps in exploring them.	
Behaviour	A possible behaviour consists in an alternate sequence (possibly infinite) of states and state transitions, where: • the state is structured in terms of components and the values of components stay unchanged between two state transitions, • the state transitions correspond to the beginning and/or the ending of actions, called events.	

	Some behaviour can be considered as more normative than some other. Thereby, behaviour can be classified as normative or non-normative according to the fact it is considered to include a few or a lot of exceptions.	
Episode	According to Regnell and Potts, on episode is a "part of a use case representing a demarcated and coherent flow of events". They help structure a use case in manageable units.	
Goal	A Goal is a future system state or behaviour to avoid, maintain, attain, cease, etc.	Intention
Message trace diagram	A Message Trace Diagram (MTD) is a graphical way of representing the communication part associated with scenarios. It exists some extensions which allow to express also internal actions which widens their scope from just expressing communication.	Sequence diagram
Open Issue	An open issue is the result of the RE process.	
Problem statement	Something that you say or write about a situation that causes difficulties.	
Product	A product is the result, which remains after the execution of a process.	
Requirement	A requirement is a change or quality criterion for some future system (version). We distinguish functional and non-functional requirements.	
Scenario	At a functional level, a scenario is a description denoting similar parts of possible behaviours limited to a subset of purposeful state components, actions and communications taking place among two or several agents. More external (richer) scenarios include information about roles, responsibilities, organisation policies, ...	Contextual scenario
Scene	All the things that are happening in a place, and the effect or situation that they cause.	
Specification	A set of behaviours of the system and of its environment.	
Use case	A use case is defined as a possibly structured set of scenarios grouped together to achieve a specific stakeholder goal. (According Jacobson) A use case is a sequence of particularised transactions banded the ones to others. The origin of these transactions	

	is the dialog between the actor and the system. A Set of these uses cases specifies all the possibilities to use the system.	

2. PROCESS

Term	Definition	Synonyms
Analyse	A cognitive activity involving the decomposition, the structuring and scoping of a knowledge as well as deducing properties of the thing, under analysis, e.g. incompleteness, incorrectness, etc. These RE-specific properties should be quite easy to list.	Understand, Reason about
Animate	The interactive process of visualising the dynamic properties associated with fragments of normative or non-normative behaviours of the composite system.	Activate, Simulate
Change	To make something or someone different.	Modify
Compare	To consider two or more process, product, requirements, etc. in order to show how they are similar to or different from each other.	
Compose	(To be composed of) to be formed of a group of parts.	Assemble, Aggregate, Integrate, Combine
Conceptualise	The process of systematically abstracting (existing or envisaged) real-world phenomena into models which highlight the essential aspects and hide the unimportant details (relative to the viewpoint taken).	Model, Abstract
Create	To make something exist that did not exist before. The process of (semi-) automatically building a product (scenario, requirements specification, etc.) in some targeted formalism, starting from a semantic definition of its content.	Compose, Design, Generate
Decompose	The process of partitioning a product/ process/ problem into more manageable units.	Atomise, Partition
Document	As opposed to ‘to conceptualise’, write down the product of activities such as analyse, compare, change, etc.	To describe, Specify, Record, Write
Elicit	The process of systematically obtaining from people new facts	Acquire,

	(scenarios, requirements) about the domain / business processes / the system under consideration.	Discover, Capture
Envision	To project what a product will be.	Project, Imagine
Explain	To make something clear or easy to understand, to give a reason for something to someone.	Clarify, Illustrate
Explore	The process of envisaging (evaluating) alternatives, or scope or pathways.	Navigate
Find	To achieve or get something that you need	Achieve, Retrieve
Gather facts	From documents.	Collect facts
Identify	To recognise and correctly name an element of a product or of a process; to perceived some coherent entity of the (existing or envisaged) real-world (or Universe of discourse) and, optionally, express it as an element of a product, process, ...	Name
Negotiate	The process of integrating different viewpoints of different stakeholders on a certain topic, in order to try to reach an agreement of all involved stakeholders. Involves mediation and reconciliation.	Mediate, Reconcile
Refine	The process (more detailed or more precise) of changing a product (or process) in a systematic way so that the changed product/process is better (more detailed or more precise) according to some characteristic than the former one.	Elaborate, Improve
Relate	The process of explicitly defining links between products between which a semantic or structural relationship exists.	Associate, Link, Structure, Map
Remove	To take something away from the place where it is.	Delete
Review	To examine, consider and judge a product or a process carefully with respect to completeness and correctness.	Assess, Evaluate
Scope	To draw the boundaries of the system.	Delimit
Search	To try to find a solution to a problem, an explanation for something.	Explore, Investigate

Select	To choose among candidates products or processes subset by carefully thinking about which is the best, most suitable, etc. for satisfying a higher-level intention.	Choose
Sort	To put things in a particular order or to arrange them in groups according to size, rank, type, etc.	Prioritise, Rank, Order, Categorise, Classify
Suggest	To provide someone with useful information with respect to the intention(s) he as to satisfy.	Advise, Recommend
Trace	To record and subsequently retrieve information about the product and process evolution in a sequential or time-ordered manner.	Record
Validate	The process of checking against stakeholders that the right product is being built.	Test, Quality assure
Verify	To get the product right.	Attest, Check
Walk through	To validate a model in a co-operative setting using a sequential process that tests components in an order. The validation is made manually, not in an automatic manner; it follows a stepwise process for checking.	Check, Validate

Bibliographie

[Aaen 92] I. Aaen, A. Siltanen, C. Sorenjsen, V. Tahvanainen, *A tale of two countries: CASE experience and expectations*, The Impact of Computer Supported Technology on Information Systems Development, North Holland Pub, Amsterdam, pp 61-93, 1992.

[Ahituv 87] N. Ahituv, A metamodel of information flow : a tool to support information systems theory. Communications of the ACM 30 (9), pp. 781-791, 1987.

[Akman 90] V. Akman, P.J.W. ten Hagen, T. Tomiyama, *A fundamental and theoretical framework for an intelligent CAD System*. Computer Aided Design, Vol. 22, No. 6, 1990.

[Alexander 79] C. Alexander, S. Ishikawa, M. Silverstein et al., *A Patron Language*. Oxford University Press, New York, 1997.

[Batini 92] C. Batini, M. Lenzerini, B. Navathe, *Conceptual database design : An entity relationship approach*. Benjamin-Cummings Publishing Company, Redwood City, 1992.

[Beck 97] K. Beck, *Smalltalk : Best Practice patrons.* Volume 1: Coding, Prentice Hall, Englewood Cliffs, NJ, 1997.

[Ben Achour 98] C. Ben Achour, C. Rolland, C. Souveyet, *A proposal for improving the quality of the organisation of scenarios collections.* Proceedings of the 4th International Workshop on Requirements Engineering Foundation for Software Quality. E. Dubois, A. Opdahl, K. Pohl (Eds). June, Pisa, Italy.

[Benali 89] K. Benali, N. Boudjlida, F. Charoy, J. C. Derniame, C. Godart, Ph. Griffiths, V. Gruhn, Ph. Jamart, D. Oldfield, F. Oquendo, *Presentation of the ALF project*, Proceedings of the International Conference on System Development Environments and Factories, 1989.

[Benjamen 99a] A. Benjamen, *Towards a dynamic construction of Process Model*, in Proceedings of the MFPE'99 Many Facets of Process Engineering Workshop, Gammarth, Tunisie, 1999.

[Benjamen 99b] A. Benjamen, *Une Approche Multi-démarches pour la modélisation des démarches méthodologiques.* Thèse de doctorat en informatique de l'Université Paris 1, 6 octobre 1999.

[Besançon 99] R. Besançon, M. Rajman, J.C. Chappelier, *Textual Similarities on a Distributional Approach*. Proceedings of the 10th International Workshop on Database and Expert Systems Applications (DEXA'99), Florence, Italy, September 1999.

[Bianco 99] G. Bianco, V. De Antonellis, S. Castano, M. Melchiori, *A Markov Random Field Approach for Querying and Reconciling Heterogeneous Databases*. Proceedings of the 10th International Workshop on Database and Expert Systems Applications (DEXA'99), Florence, Italy, September 1999.

[Boehm 88] B. Boehm, A Spiral Model of Software Development and Enhancement, IEEE Computer 21, 5, 1988.

[Booch 91] G. Booch, Object Oriented Analysis and Design with Applications, Benjamin/Cummings, 1991.

[Booch 97] G. Booch, I. Jacobson, J. Rumbaugh, *Unified Modeling Language* : version 1.0. Rational Software Coorporation, 1997.

[Bouzeghoub 90] M. Bouzeghoub, I. Comyn, *View Integration by Semantic Unification and Transformation of Data Structures*, Proceedings of the Conference on Requirements Engineering, RE'90, Lausanne, 1990.

[Brinkkemper 00] S. Brinkkemper, *Method engineering with Web-enabled methods.* In Information Systems Engineering : State of the Art and Resaerch Themes, S. Brinkkemper, E. Lindencrona, A. Solvberg (Eds.), pp. 124-133, Springer-Verlag, 2000.

[Brinkkemper 89] S. Brinkkemper, M. de Lange, R. Looman, F.H.G.C. van der Steen, *On the Derivation of Method Companionship by Meta-Modelling*. Imperial College, London, UK , 1989.

[Brinkkemper 90] S. Brinkkemper, *Formalisation of information systems modelling*, Ph. D. Thesis, University of Nijmegen, Thesis Publishers, Amsterdam, 1990.

[Brinkkemper 95] S. Brinkkemper, *Method engineering : engineering of information systems developments methods and tools*. Information & Software Technology, 37 (11) pp. 1-6, 1995.

[Brinkkemper 96] S. Brinkkemper, *Method engineering : engineering of information systems development methods and tools*, Information and Software Technology, Vol. 38, No.4, pp.275-280, 1996.

[Brinkkemper 98] S. Brinkkemper, M. Saeki, F. Harmsen, *Assembly Techniques for Method Engineering*. Proceedings of the 10th Conference on Advanced Information Systems Engineering, CAiSE'98. Pisa Italy, 8-12 June, 1998.

[Bushmann 96] F. Bushmann, R. Meunier, Rohnert et al., *Patron-Oriented Software Architecture – A System of Patrons*. John Wiley, 1996.

[Castano 92] S. Castano, V. De Antonellis, B. Zonta, *Classifying and Reusing Conceptual Schemas*. Proceedings of the 11th International Conference on Conceptual Modeling (ER'92), pp. 121-138, Karlsruhe, 1992.

[Castano 93] S. Castano, V. De Antonellis, *A Constructive Approach to Reuse of Conceptual Components*. Proceedings of Advances in Software Reuse : Selected Papers from the Second International Workshop on Software Reusability, Lucca, Italy, March 24-26, 1993. R. Prieto-Diaz,W.B. Frakes (Eds), IEEE Computer Society Press.

[Chen 91] M. Chen, J.F. Nunamaker, G. Mason, The architecture and Design of a Collaborative Environment for System Definition, DATA BASE, pp. 22-28, 1991.

[Coad 91] P. Coad, E. Yourdon, *Object-Oriented Analysis*. Yourdon Press, 1991.

[Coad 92] D. Coad, *Object oriented patterns*. Communications of the ACM, Vol. 35, No. 9, pp. 152-159, 1992.

[Coad 96] D. Coad, D. North, M. Mayliefd, *Object Models – Strategies, patterns and applications*, Yourdon Press Computing Series, 1996.

[Coplien 95] J.O Coplien, D.O. Schmidt (Eds.), *Patron Languages of Program Design*. Addison-Wesley, Reading, MA, 1995.

[Curtis 88] B. Curtis, M. Kellner, J. Over, *A Field Study of the Software Design Process for Large Systems*, Com. ACM, Vol. 31, No. 11, 1988.

[Curtis 92] B. Curtis, M. I. Kellner, J. Over : *Process Modelling*. Communications of the ACM, Vol. 35, N° 9, pp 75-90, 1992.

[Dardenne 91] A. Dardenne, S. Fickas, A. van Lamsweerde, *Goal – directed concept acquisition in requirements elicitation* . Proceedings of the 6th IEEE Workshop System Specification and Design, Como, italy, 14-21, 1991.

[De Antonnellis 91] V. De Antonnellis, B. Pernici, P. Samarati, *F-ORM METHOD : A methodology for reusing specifications*. In Object Oriented Approach i Informatin Systems, F. van Assche, B. Moulin, C. rolland (Eds.), North Holland, 1991.

[Denéckère 01] R. Denéckère, *Approche d'extension de méthodes fondée sur l'utilisation de composants génériques*. Thèse de doctorat en informatique, Université Paris 1 – Sorbonne, A paraître en janvier 2001.

[Diamantini 99] C. Diamantini, M. Panti, A Conceptual Indexing Method for Content-Based Retrieval. Proceedings of the 10th International Workshop on Database and Expert Systems Applications (DEXA'99), Florence, Italy, September 1999.

[Dowson 88] M. Dowson, *Iteration in the Software Process*, Proceedings of the 9th International Conference on Software Engineering, 1988.

[Dubois 98] E. Dubois, P. Heymans, Scenario-Based Techniques for supporting the Elaboration and the Validation of Formal Requirements, Submitted to RE Journal, 1998.

[Emmerich 91] W. Emmerich, G. Junkermann, B. Peuschel, W. Schäfer, S. Wolf, *MERLIN: Knowledge based process modelling.* 1st European Workshop on Software Process Modeling. A. Fuggetta, R. Conradi, V. Ambriola (Eds), Mila, Italy, May 1991.

[Euromethod 94] *Euromethod Architecture*, Euromethod Project, Deliverable Workpackage 3, 1994.

[Euromethod 96] *Euromethod version 1.0*, Euromethod Project, Final Deliverable, 1996.

[Finkelstein 90] A. Finkelstein, J. Kramer, M. Goedicke, *ViewPoint Oriented Software Development*, Actes de Conférence "Le Génie Logiciel et ses Applications", Toulouse, p 337-351, 1990.

[Fowler 97] M. Fowler, *Analysis Patterns : reusable objects models*, Addison-Wesley, 1997.

[Fowler 97] M. Fowler, UML distillated: applying the standard object modeling notation. Addison-Wesley, Reading, MA, 1997.

[Franckson 91] M. Franckson, C. Peugeot, *Specification of the object and process modeling language*, ESF Report n° D122-OPML-1.0, 1991.

[Franckson 94] M. Franckson, *The Euromethod deliverable model and its contribution to the objectives of Euromethod*, Proc. IFIP-TC8 Int. Conf. on Methods and Tools for the Information Systems Life Cycle, Verrijn-Stuart and Olle (eds), North-Holland, pp131-149, 1994

[Front 97] A. Front, *Dévelopement de systèmes d'information à l'aide de patrons. Application aux bases de données actives*. Thèse de doctorat à l'université Joseph Fourier – Grenoble 1, Décenmbre 1997.

[Gamma 93] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design patterns : Abstraction and Reuse of Object-Oriented Design*, Proc. of the ECOOP'93 Conf., Sringer Verlag, 1993.

[Gamma 94] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns : Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[Goldfarb 90] C. F. Goldfarb, *The SGML Handbook*, Oxford Clarendon Press, 1990.

[Graham 94] I. Graham, *Object Oriented Methods*. Addison-Wesley, 1994.

[Grosz 97] G. Grosz, C. Rolland, S. Schwer, C. Souveyet, V. Plihon, S. Si-Said, C. Ben Achour, C. Gnaho, *Modelling and Engineering the Requirements Engineering Process : an overview of the NATURE approach*. Requirements Engineering Journal 2, pp. 115-131, 1997.

[Grundy 96] J.C. Grundy, J.R.Venable, *Towards an integrated environment for method engineering*, Proc. IFIP WG 8.1 Conf. on method Engineering, Chapman and Hall, pp 45-62, 1996

[Harmsen 93] F. Harmsen, S. Brinkkemper, *Computer Aided Method Engineering based on existing Meta-CASE technology*. Proceedings of the 4th Workshop on the Next Generation of CASE Tools, Univ. Of twente, Enschede, the Netherlands, 1993.

[Harmsen 94] A.F. Harmsen, S. Brinkkemper, H. Oei, *Situational Method Engineering for Information System Projects*. In Olle T. W. and A. A. Verrijn Stuart (Eds.), Methods and Associated Tools for the Information Systems Life Cycle, Proceedings of the IFIP WG8.1 Working Conference CRIS'94, pp. 169-194, North-Holland, Amsterdam, 1994.

[Harmsen 95a] F. Harmsen, S. Brinkkemper, *Description and Manipulation of Method Fragments for Situational Method Assembly*. Proceedings of the Workshop on Management of Software Projects, Pergamon Press, London, 1995.

[Harmsen 95b] F. Harmsen, S. Brinkkemper, *Design and implementation of a method base management system for situational CASE environment*. Proc. 2nd APSEC Conference, IEEE Computer Society Press, pp 430-438, 1995.

[Harmsen 96] F. Harmsen, M. Saeki, *Comparison of four method engineering languages*, IFIP 8.1 Conference on Method Engineering, 1996, Chapman and Hall, pp 209-231, 1996

[Harmsen 97] A. F. Harmsen, Situational Method Engineering. Moret Ernst & Young , 1997.

[Haumer 98] P. Haumer, K. Pohl, K. Weidenhaupt, *Requirements Elicitation and Validation with real world scenes.* IEEE Transactions on Software Engineering, Vol. 24, N°. 12, Special Issue on Scenario Management, December. 1998.

[Henderson-Sellers 90] B. Henderson-Sellers, J.M. Edwards, *The Object-oriented Systems Life-Cycle*. Communications of the ACM (9), 1990.

[Hey 96] D. Hey, *Data Model Patrons : Conventions of Thought*. Dorset House, NY, 1996.

[Heym 92] M. Heym, H. Österle, *A reference model of information systems development*. The Impact of Computer Supported Technologies on Information Systems Development, K.E. Kendall, K. Lyytinen and J.I. DeGross (Eds.), Amsterdam, North-Holland, pp. 215-240, 1992.

[Heym 93] M. Heym, H. Osterle, *Computer-aided methodology engineering*. Information and Software Technology 35 (6/7), pp. 345-354, 1993.

[Heymans 98] P. Heymans, E. Dubois, Scenario-Based Techniques for Supporting the Elaboration and the Validation of Formal Requirements. Requirements Engineering Journal, Vol. 3, No. 3-4, 1998.

[Hidding 94] G.J. Hidding, *Methodology information : who uses it and why not?*, Proc. WITS-94, Vancouver, Canada, 1994.

[Hillegersberg 97] J. van Hillegersberg, *Metamodeling-based integration of objet-oriented systems development*. Dissertation, Thesis Publishers, Amsterdam, 1997.

[Hofstede 92] A.H.M. ter Hofstede, T.F. Verhoef, E.R. Nieuwland and G.M. Wijers, *Specification of Graphic Conventions in Methods*. Proceedings of the 3rd Workshop on Next Generation in Methods, B. Theodoulidis, A. Sutcliffe (Eds.), pp. 185-215, UMIST, Manchester, UK, 1992.

[Hofstede 93] A.H.M. ter Hofstede, *Information modelling in data intensive domains*. Dissertation, University of Nijimegen, the Netherlands, 1993.

[Hong 93] S. Hong, G. van der Goor, S. Brinkkemper, *A Formal Approach to the Comparison of Object-Oriented Analysis and Design Methodologies*. Proceedings of the 26th Hawaii International Conference on Systems Science, J. Nunamaker, R. Sprague (Eds.), Vol. 4, IEEE Computer Society Press, Los Alamitos, 1993.

[Iivari 94] J. IIvari, *Object-Oriented Information System Analysis: Comparative Analysis of six Object-Oriented Analysis Methods*. IFIP Transactions: Methods and Associated Tools for the Information System Life cycle, A. A. Verrijn-Stuart & T. W. Olle (Eds) North-Holland, 1994.

[Iivary 90] J. Iivary, *Hierarchical Spiral Model for Information Systems and Software Development*. Information and Software Technoology, Vol. 32, No. 6 and No. 7, 1990.

[Jacobson 92] I. Jacobson, M. Christerson, P. Jonsson, G. Oevergaard, *Object Oriented Software Engineering: a Use Case Driven Approach*. Addison-Wesley, 1992.

[Jacobson 95] I. Jacobson, *The Use Case Construct in Object-Oriented Software Engineering*. In John M. Carroll, editor,Scenario-Based Design: Envisioning Work and Technology in System Development, pp 309-336. John Wiley and Sons, 1995.

[Jarke 92] M. Jarke, K. Pohl, *Information systems quality and quality information systems*. Proceedings of the IFIP 8.2 Working Conference on the impact of computer-supported techniques on information systems development, Minneapolis, NM, june 1992.

[Jarke 99] M. Jarke, C. Rolland, A. Sutcliffe, R. Domges, *The NATURE requirements Engineering*. Shaker Verlag, Aachen 1999.

[Jilani 97] L.L. Jilani, R. Mili, A. Mili, *Approximate Component Retrieval : An Academic Exercise or a Practical Concern ?*. Proceedings of the 8th Workshop on Istitutionalising Software Reuse (WISR8), Columbus, Ohio, March 1997.

[Johnson 88] R.E. Johnson, B. Foote, *Designing reusable classes*. Journal of Object-Oriented Programming, Vol. 1, No. 3, 1988.

[Katayama 89] T. Katayama, *A hierarchical and functional software process description and its enaction*. Proceedings of the 11th International Conference on Software Engineering, pp. 343-352, 1989.

[Kelly 94] S. Kelly, A Matrix for a MetaCASE Environment. Information and Software Technology, 36 (6), pp. 361-371, 1994.

[Kelly 96] S. Kelly, K. Lyyttinen, M. Rossi, *Meta Edit+: A fully configurable, multi-user and multi-tool CASE and CAME environment*, Proceedings of the CAiSE'96 Conference, 20-24 may, Heraklion, Crete, Greece, Springer Verlag, 1996.

[Kinnuenen 94] K. Kinnuenen, M. Leppänen, *O/A Matrix and a Technique for Methodology Engineering*. Proceedings of the 4th International Conference on Information Systems Development, J. Zupansis and S. Wrycza (Eds.), Moderna Organizacija, Kranj, Slovenia, 1994.

[Kottemann 84] J.E. Kottemann, B.R. Konsynski, *Dynamic Metasystems for Information Systems Development.* Proceedings of the 5th International Conference on Information Systems, pp. 187-204, 1984.

[Kronlof 93] K. Kronlof, *Method Integration, Concepts and Case studies,* Wiley series in software based systems, John Wiley and sons Ltd., 1993.

[Kumar 92] K. Kumar, R.J. Welke, *"Methododlogy Engineering : A Proposal for Situation-Specific Methodologies Construction"*. In Challenges and Strategies for Research in System Development, Cotterman, W.W. and Senn, J.A. eds. , Wiley & Sons Ltd., 1992.

[Le Petit Robert 95] Le Petit Robert, French Dictionary, Dictionnaire LE ROBERT, France,1995.

[Lemaitre 95] J. Lemaitre, E. Murisasco, M. Rolbert, SgmlQL, *Un langage d'interrogation de documents SGML*. Proceedings of the 11th Conference on Advanced Data Bases, August 1995, Nancy, France.

[Lemmen 99] K.A.M. Lemmen, S. Brinkkemper, *An approach to engineering situational methods by matching project situations to existing methods.* In Proceedings ISD'98 : Evolution an challenges in System Development, Zupancic et al. (eds.), plenum Publichers, New York, pp. 247-259.

[Liu 95] H. Liu, *A visual Interface for Queryig a CASE Repository*. Proceedings of the 11th IEEE Symposium on Visual Languages (VL'95), Darmstadt, 1995.

[Lyytinen 87] Lyytinen K., *Different perspectives on information systems : problems and solutions,* ACM Computing Surveys, Vol 19, No1, 1987.

[Lyytinen 89] K. Lyytinen, K. Smolander, V-P. Tahvainen, *Modelling CASE Environments in sytems Work,* CASE'89 conference papers, Kista, Sweden, 1989.

[Maiden 98a] N.A.M. Maiden, CREWS-SAVRE: Scenarios for Acquiring and Validating Requirements. Journal of Automated Software Engineering, 1998.

[Maiden 98b] N.A.M. Maiden, M. Cisse, H. Perez, D. Manuel, *CREWS Validation Frames: Patterns for Validating Systems Requirements.* Fourth International Workshop on Requirements Engineering: Foundation for Software Quality (RESFQ), Pisa, Italy, June 8th-9th, 1998.

[Maiden 98c] N.A.M. Maiden, S. Minocha, K. Manning, M. Ryan , SAVRE: Systematic Scenario Generation and Use. International Requirements Engineering Conference (ICRE'98), Colorado Springs, Colorado, USA, April 6-10, 1998.

[Martin 92] J. Martin, J.J. *Odell, Object-Oriented Analysis & Design.* Prentice Hall, 1992.

[Marttiin 95] P. Marttiin, K. Lyytinen, M. Rossi, V-P. Tahvanainen, J-P. Tolvanen, *Modeling requirements for future CASE : issues and implementation considerations.* Information Resources Manegement Jpournal 8 (1), pp. 15-25, 1995.

[Merbeth 91] Merbeth G., *Maestro II- das intergrierte CASE-system von Softlab,* CASE systeme and Werkzeuge (Ed. H. Balzert) BI Wissenschaftsverlag, pp 319-336, 1991.

[Muller 97] P. Muller, *Modélisation objet avec UML*. Eyrolles, 1997.

[Mylopoulos 90] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis, *Telos : Representing Knowledge About Information Systems*. ACM Transactions on Information Systems, 8, 4, pp. 325-362, 1990.

[Oei 92] J.L.H. Oei, L.J.G.T. van Hemmen, E.D. Falkenberg, S. Brinkkemper, *The Meta Model Hierarchy : A Framework for Information Systems Concepts and Techniques*. University of Nijmegen, 1992.

[Oei 94] J.L.H. Oei, E.D. Falkenberg, *Harmonisation of information systems modelling and specification techniques*. In Methods and Associated Tools for the Information Systems Life Cycle, pp. 151-168, A.A. Verrijn-Stuart and T.W. Olle (Eds.), No. A-55, Elsevier Science publishers, 1994.

[Oei 95] J.L.H. Oei, *A meta model transformation approach towards harmonisation in information system modeling.* In Information Systems Concepts – towards a consolidation of views, pp. 106-127, E.D. Falkenberg, W. Hesse and A. Olivé (Eds.), Chapman & Hall, London, 1995.

[Olle 88] T. W. Olle, J. Hagelstein, I. MacDonald, C. Rolland, F. Van Assche, A. A. Verrijn-Stuart, *Information Systems Methodologies : A Framework for Understanding*, Addison Wesley (Pub.), 1988.

[OMG 97] http://www.omg.org

[Papadopoulos 99] A.N. Papadopoulos, Y. Manolopoulos, *Structure-Based Similarity Search with Graph Histograms*. Proceedings of the 10th International Workshop on Database and Expert Systems Applications (DEXA'99), Florence, Italy, September 1999.

[Plihon 94] V. Plihon, The OMT Methodology, The OOA Methodology (Coad/yourdon), The SA/SD Methodology, The Entity Relationship Methodology, The O* Methodology, The OOD Methodology, NATURE Deliverable DP2, 1994.

[Plihon 95] V. Plihon, C. Rolland, *Modelling Ways-of-Working*, Proc 7th Int. Conf. on Advanced Information Systems Engineering (CAISE), Springer Verlag (Pub.), 1995.

[Plihon 96] V. Plihon, *Un environnement pour l'ingénierie des méthodes*, Thèse de doctorat de l'Université Paris 1, janvier 1996.

[Plihon 98] V. Plihon, J. Ralyté, A. Benjamen, N.A.M. Maiden, A. Sutcliffe, E. Dubois, P. Heymans, *A Reuse-Oriented Approach for the Construction of Scenario Based Methods.* Proceedings of the International Software Process Association's 5th International Conference on Software Process (ICSP'98), Chicago, Illinois, USA, 14-17 June 1998.

[Poels 00] G. Poels, S. Viaene, G. Dedene, *Distance Mesure for Information System Reengineering*. Proceedings of the 12th Conference on Advanced Information Systems Engineering CAISE'00, Stockholm, Sweden, June 2000.

[Poels 00a] G. Poels, G. Dedene, Distance-based software mesurrement : necessary and sufficient properties for software measures. Information and Software Technology, 42, pp. 35-46, 2000.

[Potts 89] C. Potts, *A Generic Model for Representig Design Methods.* Proceedings of the 11th International Conference on Software Engineering, 1989.

[Potts 94] C. Potts et al., *Inquiry-based Requirements Analysis*, IEEE Software, Mars 1994.

[Prakash 94] N. Prakash, *A Process View of Methodologies*, 6th Int. Conf. on Advanced Information Systems Engineering, CAISE'94, Springer Verlag, 1994.

[Prakash 97] N. Prakash, *Towards a formal definition of methods*, in Requirements Engineering, Vol. 2 , N° 1, 1997.

[Prakash 99] N. Prakash, *On Method Statics and Dynamics*. Information Systems Vol. 24, No. 8, pp. 613-637, 1999.

[Prat 97] N. Prat, *Goal formalisation and classification for requirements engineering*. Proceedings of the Third International Workshop on Requirements Engineering: Foundations of Software Quality REFSQ'97, Barcelona, pp. 145-156, June 1997.

[Prat 99] N. Prat, *Réutilisation de la trace par apprentissage dans un environnement pour l'ingénierie des processus*, Thèse de doctorat en informatique de l'université Pris 1, Le 3 février 1999.

[Pree 95] W. Pree, Design Patterns for Object-Oriented Software Development. Addison Wesley, 1995.

[Prieto-Diaz 87] R. Prieto-Diaz, P. Freeman, *Classifying Software for reusability*, IEEE Software, Vol. 4, N° 1, January 1987.

[Protsko 89] L.B. Protsko, P.G. Sorenson, J.P. Tremblay, Mondrian/ system for automatic generation of dataflow diagrams. Information and Software technology, 31 (9), pp/ 456-471, 1989.

[Punter 96] H.T. Punter, K. Lemmen, "The MEMA model : Towards a new approach for Method Engineering". Information and Software Technology, Vol. 38, No.4, pp.295-305, 1996.

[Ralyté 97] J. Ralyté, C. Ben Achour, *Scenario Integration into Requirements Engineering Methods.* Proceedings of the Workshop on the Many Facets of Process Engineering (MFPE'97), Gammarth, Tunis, September 22-23, 1997.

[Ralyté 99a] J. Ralyté, C. Rolland, V. Plihon, *Method Enhancement by Scenario Based Techniques*. Proceedings of the 11th Conference on Advanced Information Systems Engineering, Heidelberg, Germany, June 14-18, 1999.

[Ralyté 99b] J. Ralyté, *Reusing Scenario Based Approaches in Requirement Engineering Methods: CREWS Method Base*. Proceedings of the First International Workshop on the Requirements Engineering Process - Innovative Techniques, Models, Tools to support the RE Process, Florence, Italy, September 1999.

[Rieu 99] D. Rieu, J-P. Giraudin, *Patrons Orintés-objet*. Edition Hermès, 1999.

[Rolland 00a] C. Rolland, S. Nurcan, G. Grosz, *A Decision Making Pattern for Guiding the Enterprise Knowledge Development Process*. Journal of Information and Software Technology, Elsevier, 42, pp. 313-331, 2000.

[Rolland 00b] C. Rolland, J. Stirna, N. Prekas, P. Loucopoulos, G. Grosz, *Evaluating a Pattern Approach as an Aid for the Development of Organisational Knowledge : An Empirical Study*. Proceedings of the 12th Conference on Advanced Information Systems Engineering (CAISE 2000), Stockholm, Sweden, May 2000.

[Rolland 93] C. Rolland, *Modeling the Requirements Engineering Process*, Information Modelling and Knowledge Bases, IOS Press, 1993

[Rolland 94a] C. Rolland, *Modeling the evolution of artifacts*, 1st IEEE International Conference on Requirements Engineering, Colorado Springs, Colorado, 1994.

[Rolland 94b] C. Rolland, G. Grosz, *A General Framework for Describing the Requirements Engineering Process*, C. IEEE Conference on Systems Man and Cybernetics, CSMC94, San Antonio, Texas, 1994.

[Rolland 94c] C. Rolland, N. Prakash, *A Contextual Approach to modeling the Requirements Engineering Process*, SEKE'94, 6th International Conference on Software Engineering and Knowledge Engineering, Vilnius, Lithuania, 1994.

[Rolland 95] C. Rolland, C. Souveyet, M. Moreno. *An Approach for Defining Ways-Of-Working*, in the Information Systems Journal, 1995.

[Rolland 96a] C. Rolland, V. Plihon, *Using generic chunks to generate process models fragments*, Proc.of 2nd IEEE International Conference on Requirements Engineering", ICRE'96, Colorado Spring, 1996.

[Rolland 96b] C. Rolland, N. Prakash, *A proposal for context-specific method engineering*, IFIP WG 8.1 Conference on Method Engineering, Chapman and Hall, pp 191-208, Atlanta, Gerorgie, USA, 1996.

[Rolland 97] C. Rolland, A Primer for Method Engineering, Actes de conférence INFORSID'97, Toulouse, Juin 1997.

[Rolland 98a] C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyté, A. Sutcliffe, N.A.M. Maiden, M. Jarke, P. Haumer, K. Pohl, Dubois, P. Heymans, *A Proposal for a Scenario Classification Framework*. Requirements Engineering Journal 3 :1, 1998.

[Rolland 98b] C. Rolland, C. Souveyet, C. Ben Achour, *Guiding Goal Modelling Using Scenarios*. IEEE Transactions on Software Engineering, special issue on Scenario Management, Vol. 24, No. 12, 1055-1071, Dec. 1998.

[Rolland 98c] C. Rolland, *A Comprehensive View of Process Engineering*. Proceedings of the 10th International Conference CAiSE'98, B. Lecture Notes in Computer Science 1413, Springer Verlag Pernici, C. Thanos (Eds), Pisa, ITALY, June 1998.

[Rolland 98d] C. Rolland, V. Plihon, J. Ralyté, *Specifying the reuse context of scenario method chunks*. Proceedings of the 10th Conference on Advanced Information Systems Engineering, CAiSE'98. Pisa Italy, 8-12 June, 1998.

[Rolland 98e] C. Rolland, C. Ben Achour, *Guiding the construction of textual use case specifications*. Data & Knowledge Engineering Journal Vol. 25 N° 1, pp. 125-160, (ed. P. Chen, R.P. van de Riet) North Holland, Elsevier Science Publishers. March 1998.

[Rolland 99] C. Rolland, N. Prakash, A. Benjamen, *A multi-model view of process modelling*. Requirements Engineering Journal, p. 169-187,1999.

[Rossi 95] M. Rossi, S. Brinkkemper, *Metrics in Method Engineering.* Proceedings of the 7th International Conference on Advanced Information Systems Engineering CAISE'95, Jyvaskyla, Finland, June 1995.

[Rossi 96] M. Rossi, S. Brinkkemper, *Complexity Metrix for Systems Development Methods and Techniques*. Information Systems, Vol. 21, No. 2, pp. 209-227, 1996.

[Royce 70] W. W. Royce, *Managing the development of large software systems*; Proceedings of the IEEE WESCON, August, 1970.

[Rumbaugh 91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[Russo 95] Russo et al, *The use and adaptation of system development methodologies*, Proc. 1995 Intl. Resources Management. Association Conference, Atlanta, 1995.

[Saeki 91] M. Saeki, T. Kaneko, M. Sakamoto, *A method for software process modelling and description using LOTOS*, Proc. 1st Intl. Conference on the Software Process, IEEE Computer Society Press, Los Alamitos, CA, USA, pp 90-104, 1991.

[Saeki 93] M. Saeki, K. Iguchi, K Wen-yin, M Shinohara, A meta-model for representing software specification & design methods. Proc. of the IFIP¨WG8.1 Conference on InformationSystems Development Process, Come, pp 149-166, 1993.

[Saeki 94a] M. Saeki, K. Wen-yin, Specifying Software Specification and Design Methods. Proceedings of Conference on Advanced Information Systems Engineering, CAISE'94, Lecture Notes in Computer Science 811, Springer Verlag, pp. 353-366, Berlin, 1994.

[Saeki 94b] M. Saeki, K. Wenyin, *PCTE based Tool for Supporting Collaborative Specification Development*, Conference PCTE'94, San Francisco, December 1994.

[Schneider 99] J.P. Schneider, J.P. Winters, *Applying Use Cases : A Practival Guide*. Addison-Wesley, 1999.

[Seligmann 89] P. S. Seligmann, G. M. Wijers, H. G. Sol, *Analyzing the structure of I. S. methodologies, an alternative approach*, in Proc. of the 1st Dutch Conference on Information Systems, Amersfoort, The Netherlands, 1989.

[Shlaer 88] S. Shlear, S. J. Mellor, *Object Oriented Systems Analysis*. Prentice-Hall, 1988.

[Shlaer 92] S. Shlear, S. J. Mellor, *Object Lifecycles. Modeling the World in States*. Prentice-Hall, 1992.

[Si-said 96] S. Si-said, G. Grosz, C. Rolland, *Mentor, A computer aided Requirements Engineering Environment*, Proceedings of the 8th CAISE Conf. Chalenges In Modern Information Systems, Heraklion, Crete, Greece, May 1996.

[Si-Said 99] S. Si-Said, *Proposition pour la modélisation etle guidage des processus d'analyse des systèmes d'information*. Thèse de doctorat en informatique de l'université Paris 1. Février 1999.

[Smolander 91] K. Smolander, K. Lyytinen, V. Tahvanainen, P. Marttiin, *MetaEdit – A Flexible Graphical Environment for Methodology Modelling*. Proceedings of the 3th International Conference in Advanced Information Systems Engineering CAISE'91, Trondheim, Norway, May 1991.

[Smolander 91] K. Smolander, K. Lyytinen, V.-P. Tahvanainen, P. Marttiin, *Meta-Edit - A flexible graphical environment for methodology modelling*, Advanced Information Systems Engineering, (Eds. Andersen R., Bubenko J. and Solvberg A.), LNCS#498, Springer-Verlag, 1991.

[Smolander 92] K. Smolander, *OPRR – A Model for Methodology Modeling*. Next Generation of Communication Systems, IOS press, pp. 224-239, 1992.

[Sommerville 87] I. Sommerville, R. Welland, S. Beer, *Describing software design methodologies*. The Computer Journal Vol. 30, No. 2, pp. 128-133, 1987.

[Song 92] X. Song, L.J. Osterweil, *Towards objective, systematic, design-method comparison*. IEEE Software, Vol. 34, No.5, May, pp. 43-53, 1992.

[Song 95] X. Song, *A Framework for Understanding the Integration of Design Methodologies*. In: ACM SIGSOFT Software Engineering Notes, Vol. 20, N°1, pp. 46-54, 1995.

[Sorenson 88] P.G. Sorenson, J.P. Tremblay, A.J. McAllister, *The Metaview System for Many Specificatio Environments*. IEEE Software, pp. 30-38, 1988.

[Souveyet 97] C. Souveyet, R. Deneckere, C. Rolland, *State of art : a comparaison of six oriented analysis methods*. TOOBIS project, deliverable T23D1.2, part 1, 1997.

[Souveyet 98] C. Souveyet, M. Tawbi, *Process Centred Approach for Developing : Tool Support of Situated Methods.* Proceedings of the Ninth International DEXA Conference on Database and Expert Systems Applications, University of Vienna, Austria, 24-28 August, 1998.

[Sowa 92] J.F. Sowa, J.A. Zachmen, *Extending and formalising the framework for information systems architecture*. IBM Systems Journal, Vol. 31, No. 3, pp. 590-616, 1992.

[Sutcliffe 98a] A. G. Sutcliffe, *Scenario-based Requirements Analysis*. Requirements Engineering Journal, Vol (3) N° 1, (ed. P. Loucopoulos, C. Potts), Springer Verlag. 1998.

[Sutcliffe 98b] A.G. Sutcliffe, N.A.M. Maiden, S. Minocha, D. Manuel, *Supporting Scenario-based Requirements* Engineering. IEEE Transactions on Software Engineering: Special Issue on Scenario Management, Vol. 24, No. 12, 1998.

[Sutcliffe 99] A. G. Sutcliffe, J. Galliers and S. Minocha, *Human Errors and System Requirements*. Fourth IEEE International Symposium on Requirements Engineering (RE'99).

[Tawbi 00] M. Tawbi, F. Velez, C. Ben Achour, C. Souveyet, *Scenario Based RE with CREWS-L'Ecritoire : Experimenting the approach.* RFSQ'00.

[Tawbi 99a] M. Tawbi, C. Souveyet, *Guiding Requirement Engineering with a Process Map*, Proceedings of MFPE'99 : 2nd International Workshop on the Many Facets of Process Engineering, Gammarth, Tunisia, 12-14, May 1999.

[Tawbi 99b] M. Tawbi, C. Ben Achour, F. Velez, *Guiding the Process of Requirements Elicitation trough Scenario Analysis : Results of an Empirical Study*. Proceedings of the First International Workshop on the Requirements Engineering Process - Innovative Techniques, Models, Tools to support the RE Process, Florence, Italy, September 1999.

[Tolvanen 93] J.P. Tolvanen, K. Lyytinen, *Flexible method adaptation in CASE environements – The metamodeling approach.* Skandinavian Jurnal of Information Systems. Vol.5, No.1, pp. 51-77, 1993.

[Tomiyama 89] T. Tomiyma, T. Kiriyama, H. Takeda, D. Xue, H. Yoshikaya, *Metamodel : A Kaey to Intelligent CAD Systems*. Research in Engineering Design, Vol. 1, pp. 19-34. 1989.

[UML 00] Rational Software Corporation, *Unified Modelling Language version 1.3*. Available at http://www.rational.com/uml/resources/documentation/, 2000.

[Van Slooten 93] K. van Slooten, S. Brinkkemper, *A Method Engineering Approach to Information Systems Development.* In Information Systems Development process, N. Prakash, C. Rolland, B. Pernici (Eds.), Elsevier Science Publishers B.V. (North-Holand), Amsterdam, 1993.

[Van Slooten 96] K. van Slooten, B. Hodes, *Characterising IS development project*, IFIP WG 8.1 Conference on Method Engineering, Chapman and Hall, pp 29-44, 1996

[Venable 93] G.R. Venable, *CoCoA: a conceptual data modelling approach for complex problem domains*, Ph.D. dissertation, SUNY, Binghampton, 1993

[Verhoef 95] T.F. Verhoef, A.H.M. ter Hofstede, *Feasibility of Flexible Information Modelling Support*. Advanced Informtion Systems Engineering, J. Iivari, K. Lyytinen and M. Rossi (Eds.), Springer-Verlag, pp. 168-185, 1995.

[Vlissides 96] J.M. Vlissides, J.O. Coplien, N.L. Kerth (Eds.), *Patron Languages of program Design 2.* Addison-Wesley, 1996.

[Welke 91] R.J. Welke, K. Kumar, *Method engineering : a proposal for situation-specific methodology construction*. Systems Analysis and Design :A Research Agenda, Cotterman and Senn (Eds.), Wiley, 1991.

[Welke 92a] R.J. Welke, K. Kumar, *Method Engineering : A Proposal for Situation-specific Methodology Construction*, in Systems Analysis and Design : A Research Agenda, Cotterman and Senn(eds), Wiley, pp257-268, 1992.

[Welke 92b] R.J. Welke, *The CASE Repository : More than another database application*. In Challenges and Strategies for Research in Systems Development. W.W.Ctterman and J.A. Senn (Eds.), Wiley, Chichester UK, 1992.

[Wijers 90] G. M. Wijers, H.E. Van Dort, *Experiences with the use of CASE tools in the Netherlands*, Advanced Information Systems Engineering, pp 5-20, 1990.

[Wijers 91] G. M. Wijers, *Modeling Support in Information Systems Development*, PhD Thesis, Thesis Publishers Amsterdam, 1991.

[Wirfs-Brock 90] J. Wirfs-Brock, H.E. van Dort, *Experiences with the use of CASE tools in the Netherlands*. Advanced Information Systems Engineering, pp. 5-20, 1990.

[Wynekoop 93] J. Wynekoop, N. Russo, *System development methodologies:unanswered questions and the research-practice gap*, in Proc.14th Intl. Conf. Inf. Syst., New York, ACM Pub. pp 181- 190, 1993.

[Yourdon 92] E. Yourdon, *The decline and fall of the american programmer*, Prentice Hall, Englewood Cliffs, NJ, 1992.

[Yu 94] E. Yu, J. Mylopoulos, *From ER to AR_modelling strategic Actor Relationships for Business Process Reengineering* . In Proceedings of the 13th International Conference on the Entity-Relatioship Approach – ER'94, Manchester (UK), December 13-16, 1994.