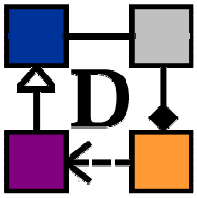
# Analyse des diagrammes de l'apprenant dans un EIAH pour la modélisation orientée objet - Le système ACDC

Ludovic Auxepaules



**Thèse de doctorat de l'Université du Maine**

*spécialité*

Informatique

*présentée par*

**Ludovic Auxepaules**

*pour l'obtention du titre de*

Docteur de l'Université du Maine

# Analyse des diagrammes de l'apprenant dans un EIAH de la modélisation orientée objet

## Le système ACDC

***Soutenue publiquement le 24 septembre 2009 devant le jury composé de :***

***Président :***

| | |
|---|---|
| M. Jean-Marc LABAT | Professeur à l'Université Pierre et Marie Curie, Paris 6 |

***Rapporteurs :***

| | |
|---|---|
| M. Thierry NODENOT | Professeur à l'Université de Pau et des Pays de l'Adour |
| M. Jean-François NICAUD | Professeur à l'Université Joseph Fourier, Grenoble |

***Examinateurs :***

| | |
|---|---|
| M. Xavier LE PALLEC | Maître de conférences à l'Université de Lille |
| M. Christophe CHOQUET | Professeur à l'Université du Maine |
| M$^{me}$. Dominique PY | Professeure à l'Université du Maine<br>Directrice |

Thèse préparée au sein du Laboratoire d'Informatique de l'Université du Maine Le Mans
Faculté des Sciences - Université du Maine – Avenue Olivier Messiaen, 72085 - Le Mans Cedex 9

# *Remerciements*

*Ma gratitude va à toutes les personnes que j'ai eu l'occasion de côtoyer et de rencontrer lors de ces quelques années de thèse. Chacune d'entre elles a participé à sa manière à l'aboutissement de cette thèse. Je prie à toutes celles que je n'aurais pas mentionnées de m'en excuser.*

*Je remercie tout d'abord Jean-Marc Labat, Professeur à l'Université Pierre et Marie Curie Paris 6, de m'avoir fait l'honneur de présider le jury de ma soutenance de thèse.*

*Je tiens à remercier Jean-François Nicaud, Professeur à l'Université Joseph Fourier de Grenoble, et Thierry Nodenot, Professeur à l'Université de Pau et des Pays de l'Adour, de m'avoir fait l'honneur d'accepter d'être rapporteurs de ma thèse.*

*Je remercie Christophe Choquet, Professeur au Laboratoire d'Informatique de l'Université du Maine (LIUM) et Xavier Le Pallec, Maître de Conférences à l'Université de Lille, d'avoir accepté de faire partie de mon jury de thèse.*

*Je tiens à remercier Dominique Py, Professeure au LIUM, d'avoir dirigé cette thèse, de m'avoir conseillé et guidé tout au long de mes travaux.*

*Je tiens également à remercier Mathilde Alonso et Thierry Lemeunier sans qui l'environnement Diagram et ma recherche ne seraient pas ce qu'ils sont. Un grand merci pour les travaux que nous avons menés conjointement depuis le début de cette thèse.*

*J'adresse tous mes remerciements aux membres du personnel du LIUM et du département d'informatique devenus ensuite mes collègues. J'ai pu réaliser grâce à eux dans de bonnes conditions ma formation universitaire et ensuite ma thèse et mes enseignements : merci à Pierre, Christophe, Phillipe, Jérome, Arnaud, Paul, Bruno, Étienne, Edith, Martine et tous les autres. Je remercie également mes collègues de Laval que j'ai pu rencontrer à différentes occasions : Pierre, Claudine, Philippe, Lahcen…*

*Une thèse ne serait rien sans un environnement de travail agréable. J'en profite donc pour remercier les doctorantes de mon bureau : Bérangère, Christelle, Carole, Aina et Mathilde. Je remercie également les autres doctorants du LIUM qui vivent ou ont vécu cette expérience qu'est une thèse et plus particulièrement Naïma, Julie, Patricia, Boubekeur, Noa, Hassina, Firas, Thierry, Antoine, Richard, Vincent et Anthony.*

*Une thèse, c'est aussi des rencontres toujours enrichissantes notamment lors des manifestations liées à la recherche telles que les conférences : je remercie notamment Elisabeth, Anne, Vanda, Monique, Françoise, Hélène, Xavier, Jean-Phillippe, Jean-Marc, Éric, Pierre-André, Jean-Hugues, Richard, Ivan et bien d'autres pour leur intérêt porté à l'égard de mes travaux, leurs conseils et leurs encouragements. Merci également à tous les doctorants rencontrés dans ces mêmes occasions et avec qui avec j'ai pu discuter de nos préocupations communes : Nabila, Fatoumata, Mariam, Amine, Mohamedade, Douadi, Tom…*

*Je n'oublie pas tous mes étudiants, et en particulier les promotions de Master 1ère année d'informatique 2008-2009 et de DEUST ISR 2ème année 2009-2010. Je remercie les étudiants des différentes promotions de DEUST 1ère année ISR ayant utilisé Diagram et ainsi permis de mener mes travaux sur leurs diagrammes très variés. Je remercie le bureau et les membres de l'association Vitamime pour leur bonne humeur et leur participation à la vie de l'« IUP MIME ».*

*Je remercie Alexandre pour son amitié depuis le lycée, le soutien et le partage de nos expériences respectives dans le monde de l'enseignement et de la recherche dans deux domaines différents.*

*Je remercie toute ma famille pour son soutien. Un grand merci en particulier à mon oncle Pascal qui, lorsqu'il était à ma place et moi enfant, a éveillé ma curiosité avec ses petits robots et ses gros écrans. Merci à mes parents et à mon frère pour leur amour et leur soutien dans toutes les étapes que j'ai traversées.*

*Je terminerai cette page en remerciant Typhaine, ma compagne, de m'avoir aidé, soutenu, réconforté et supporté (dans tous les sens du terme) tout au long de cette thèse.*

# RÉSUMÉ

Nos travaux s'inscrivent dans le cadre des recherches menées sur les **E**nvironnements **I**nformatiques pour l'**A**pprentissage **H**umain (EIAH) et plus particulièrement dans celui du projet **I**nteraction et **C**onnaissance (I&C) du **L**aboratoire d'**I**nformatique de l'**U**niversité du **M**aine (LIUM). Le projet I&C vise à élaborer des modèles, méthodes et outils pour la conception d'environnements dédiés à l'apprentissage particulier de la modélisation. Ce projet est appliqué actuellement à l'apprentissage des concepts de la **M**odélisation **O**rientée **O**bjet (MOO) et a suscité le développement d'un EIAH nommé *Diagram*.

Nous abordons dans nos travaux de thèse le problème d'analyse des réponses de l'apprenant lors de l'activité de modélisation et en particulier lors de l'activité d'initiation à la construction d'un diagramme de classes UML de niveau analyse à partir de spécifications textuelles. Dans ce contexte, il n'est pas possible de mettre en place un résolveur pédagogique.

L'objectif principal de cette thèse est de proposer une méthode pour l'analyse automatique des diagrammes de l'apprenant dans le cadre de l'activité de modélisation en informatique menée par des novices. Cette méthode doit être conçue à un niveau indépendant des besoins pédagogiques, afin de garantir une certaine généricité, tout en demeurant assez efficace pour que les résultats puissent servir à la production de rétroactions synchrones dans un EIAH (notamment dans l'environnement *Diagram*). Pour répondre à cet objectif, nous avons étudié conjointement les environnements existants dans le cadre de l'apprentissage de la modélisation et de l'analyse des productions des apprenants, les caractéristiques intrinsèques des modèles à analyser et enfin des travaux relatifs à la restructuration, la transformation et l'appariement de modèles. À partir de cette étude, nous avons retenu le principe de concevoir un outil de diagnostic basé sur la comparaison et l'appariement des constituants de plusieurs diagrammes. Cette proposition s'inspire des concepts et des techniques d'appariement de modèles et se concentre principalement sur les aspects structurels des modèles à apparier. Pour qualifier les écarts entre les éléments des modèles, nous nous focalisons sur la notion de différences (exprimées dans une taxonomie que nous avons définie) entre le diagramme de l'apprenant et un diagramme de référence construit par un expert. L'approche a été appliquée pour analyser les diagrammes de classes UML mais peut être généralisable à d'autres types de modèles structurés.

Nous avons instancié notre méthode d'appariement sous forme d'un composant logiciel, nommé ACDC (***A**utomatic **C**lass **D**iagrams **C**omparator*) que nous avons ensuite intégré dans *Diagram*. ACDC respecte les normes et les standards actuels de la MOO en utilisant notamment le composant UML2 du projet EMP (***E**clipse **M**odeling **P**roject*) et le format XMI (***XML M**etadata **I**nterchange*). Les résultats d'ACDC (les différences relevées) sont traités dans *Diagram* pour la production de rétroactions pédagogiques synchrones destinées à l'apprenant.

Nous avons évalué la pertinence et la qualité des résultats produits par ACDC en dehors de *Diagram* à l'aide de quatre mesures de qualité (Précision, Rappel, F-Mesure et *Recall*) sur un corpus de quatre-vingt-deux diagrammes que nous avons collectés dans des situations réelles d'apprentissage. Dans le cadre de l'environnement *Diagram* (avec le diagnostic et la production des rétroactions pédagogiques), une expérimentation en situation réelle d'apprentissage a été menée fin 2008.

# TABLE DES MATIÈRES

# INTRODUCTION

Avec l'avènement des méthodes utilisées en génie logiciel et du langage UML (***U**nified **M**odeling **L**anguage*), la modélisation et plus particulièrement la modélisation orientée objet ont pris depuis une dizaine d'années une place croissante dans l'activité des informaticiens, dans le cadre des projets de développement logiciel. La MOO (**M**odélisation **O**rientée **O**bjet) est ainsi enseignée dans la majorité des cursus universitaires en informatique. Elle vise à appréhender les concepts du paradigme orienté objet et la construction de modèles dans les différentes phases du cycle de développement d'un logiciel. La modélisation est une activité ouverte, pour laquelle il n'existe pas de méthode générale de résolution. Elle requiert la maîtrise des concepts orientés objet, ainsi que des stratégies efficaces qui sont acquises par la pratique et l'entraînement sur des exemples. Dans le cadre de nos travaux, nous nous intéressons plus particulièrement à l'une des activités fondamentales intervenant dans l'enseignement de la modélisation orientée objet : la construction à partir de spécifications textuelles (sous forme d'un énoncé) d'un diagramme de classes UML de niveau analyse par des étudiants novices.

Actuellement, peu de chercheurs se sont penchés sur les problématiques concernant l'apprentissage de la modélisation avec des EIAH (**E**nvironnements **I**nformatiques pour l'**A**pprentissage **H**umain) et sur le cas plus précis de l'apprentissage de la modélisation orientée objet de niveau analyse. Néanmoins, des environnements informatiques dédiés à l'apprentissage de la modélisation ont été développés ces dernières années : ModelsCreator [Politis *et al.* 2001], KERMIT [Suraweera & Mitrovic 2002], Modellingspace [Komis *et al.* 2003], DesignFirst-ITS (autrement appelé CIMEL-ITS) [Moritz & Blank 2005], Collect-UML [Baghaei & Mitrovic 2005] et StudentUML [Ramollari & Dranidis 2007]. Plusieurs approches ont été explorées (des micromondes, des systèmes tutoriels intelligents et des collecticiels). Certaines approches se concentrent plus particulièrement sur l'interaction, portent un jugement sur les actions de l'apprenant et lui proposent des explications. D'autres mettent davantage l'accent sur le fait que l'activité de modélisation se déroule en groupe : ils supportent et encouragent la collaboration entre les apprenants pendant l'activité de modélisation. Ce domaine, relativement difficile et peu exploré, pose encore de nombreuses questions de recherche.

Le projet I&C (**I**nteraction et **C**onnaissance) du LIUM (**L**aboratoire d'**I**nformatique de l'**U**niversité du **M**aine) dans lequel s'inscrit notre thèse aborde la conception des EIAH de la modélisation en mettant l'accent sur le lien entre intéraction et connaissance. Ce projet vise à élaborer des modèles, méthodes et outils pour la conception d'environnements dédiés à l'apprentissage de la modélisation, en se fondant sur le postulat selon lequel la connaissance émerge de l'interaction. Le projet est actuellement appliqué à l'apprentissage des concepts de la modélisation orientée objet par des novices, pour lequel un EIAH spécifique, appelé Diagram, a été conçu et développé.

Le problème général sur lequel nous nous focalisons est celui de l'analyse des productions de l'apprenant. Ce problème a suscité de nombreux travaux en EIAH, notamment dans les domaines scientifiques tels que l'algèbre, la géométrie, la physique, où il est généralement possible d'utiliser des résolveurs pédagogiques. Mais le domaine particulier de la modélisation amène des questions nouvelles : comment valider une réponse de l'apprenant en l'absence de résolveur pédagogique? Quelles rétroactions peuvent être fournies? Comment répondre à ces questions en conservant souplesse et généricité?

L'objectif principal de cette thèse est de proposer une méthode pour l'analyse automatique des diagrammes de l'apprenant dans le cadre de l'activité de modélisation menée par des novices. Cette méthode doit être conçue à un

niveau indépendant des besoins pédagogiques, afin de garantir une certaine généricité, tout en demeurant assez efficace pour que les résultats puissent servir à la production de rétroactions synchrones dans un EIAH (notamment dans l'environnement Diagram). Pour répondre à cet objectif, nous avons étudié conjointement les environnements existants dans le cadre de l'apprentissage de la modélisation et de l'analyse des productions des apprenants, les caractéristiques intrinsèques des modèles à analyser et enfin des travaux relatifs à la restructuration, la transformation et l'appariement de modèles. À partir de cette étude, nous avons retenu le principe de concevoir un outil de diagnostic basé sur la comparaison et l'appariement des constituants de plusieurs diagrammes. Cette proposition s'inspire des concepts et des techniques d'appariement de modèles et se concentre principalement sur les aspects structurels des modèles à apparier. Pour qualifier les écarts entre les éléments des modèles, nous nous focalisons sur la notion de différences entre le diagramme de l'apprenant et un diagramme de référence construit par un expert. L'approche a été appliquée pour analyser les diagrammes de classes UML mais peut être généralisable à d'autres types de modèles structurés.

Pour décrire nos travaux, nous avons organisé ce document en sept parties.

La première partie décrit les notions et caractéristiques propres au domaine d'apprentissage de la modélisation orientée objet et à son enseignement. Nous définissons tout d'abord ce que sont les modèles, la modélisation puis le génie logiciel et sa part de modélisation pour ensuite nous concentrer sur la modélisation orientée objet, le langage UML et les diagrammes de classes UML. Nous concluons ce chapitre par un exemple d'exercice de construction d'un diagramme de classes destiné à des novices et une analyse de plusieurs représentations sous forme de graphes de ce même exemple.

La seconde partie est un état de l'art des environnements informatiques dédiés à l'apprentissage de la modélisation. Nous définissons en premier lieu leurs caractéristiques, leurs objectifs communs et ensuite leurs spécificités propres. Ces spécificités ont trait à la vérification de la cohérence des modèles, l'analyse des réponses de l'apprenant, la production de rétroactions pédagogiques, le suivi des apprenants et enfin la favorisation de l'activité collective de modélisation. Nous présentons en particulier l'EIAH Diagram qui est le support de nos travaux.

La troisième partie expose les approches classiques d'analyse des productions des apprenants dans les EIAH et plus particulièrement dans le domaine de la modélisation. Nous faisons ici ressortir la manière dont ces approches permettent d'analyser les réponses, dans quel contexte elles s'appliquent et quels sont leurs avantages et leurs limites.

La quatrième partie précise les concepts inhérents au problème d'appariement de modèles, leurs contextes d'application et les techniques classiques employées pour y répondre. Nous clarifions tout d'abord les concepts et la terminologie relatifs à l'appariement de modèles. Nous présentons ensuite une classification des différentes techniques individuelles d'appariement et la manière dont elles peuvent être combinées dans un système d'appariement. Nous complétons enfin ce chapitre en présentant quelques exemples d'approches existantes d'appariement de modèles et la manière dont elles abordent et proposent une solution à ce problème.

La cinquième partie concerne la méthode proposée pour la comparaison et l'appariement de diagrammes de classes, nommée ACDC (***A**utomatic **C**lass **D**iagrams **C**omparator*). Nous commençons par exposer en premier lieu les spécificités de notre problème d'appariement découlant du contexte d'apprentissage qui ont orienté les choix retenus

dans notre proposition. Ensuite, nous présentons le fonctionnement général de notre méthode et la détaillons en suivant le processus d'appariement des entrées vers les résultats produits. Nous précisons tout d'abord les modèles et le prétraitement réalisé sur ces derniers. Ensuite, nous définissons le processus d'appariement et plus particulièrement la mesure de similarité que nous avons utilisée pour comparer les constituants des diagrammes (i.e. prise en compte des dimensions descriptives des éléments et de leur structuration dans les modèles à comparer) et la mécanique de sélection et de construction itérative de l'appariement final. Enfin nous précisons comment nous qualifions les appariements à l'aide d'une taxonomie des différences que nous avons définie. Elle se base sur la structure et la sémantique générale des modèles comparés.

La sixième partie présente l'application de notre méthode d'appariement au diagnostic des productions de l'apprenant et à la production de rétroactions pédagogiques. Cette partie fait le lien entre nos travaux et ceux de Mathilde Alonso sur l'interaction et la production d'aides pédagogiques à l'apprenant dans Diagram [Alonso 2009] [Alonso *et al.* 2008]. Nous décrivons tout d'abord une méthode de vérification de la syntaxe du diagramme construit par l'apprenant puis le lien entre nos travaux et ceux de Mathilde Alonso pour la production de rétroactions pédagogiques. Nous concluons cette partie en présentant un exemple complet de diagnostic et de production de rétroactions.

La dernière partie décrit l'implémentation effective de nos travaux dans Diagram : le système ACDC. Nous présentons ensuite la méthode et les résultats de notre évaluation de la pertinence et des performances du système ACDC réalisée hors-ligne sur les diagrammes construits par des étudiants en situation réelle d'apprentissage pour trois exercices de complexité croissante et ayant des spécificités propres.

Nous concluons ce mémoire en mettant en avant les apports, les limites et les perspectives des propositions de nos travaux.

# CHAPITRE 1

# La modélisation orientée objet et son enseignement

# PLAN DU CHAPITRE

Nos travaux s'inscrivent dans le cadre de l'apprentissage de la modélisation orientée objet lors de l'activité de construction d'un diagramme de classes par des novices. La modélisation et les modèles orientés objet sont propres au domaine de l'informatique. Nous allons présenter de manière générale les modèles et la modélisation, puis nous focaliser successivement sur ceux de la modélisation dans le paradigme objet en génie logiciel, sur le langage UML (***U**nified **M**odeling **L**anguage*) devenu le langage de référence de la modélisation orientée objet et enfin sur un type précis de modèle UML nous intéressant, le diagramme de classes.

Dans ce chapitre, nous commençons par définir ce que sont les modèles, la modélisation et l'activité de modélisation de manière générale puis dans le contexte de l'informatique où les notions de langages et de métamodèle apparaissent. Nous précisons ensuite les concepts fondamentaux du paradigme objet en génie logiciel où la modélisation orientée objet est utilisée et les méthodes couramment employées pour enseigner ces concepts. Nous nous concentrons après sur le langage UML devenu le langage de la modélisation et de la conception orientées objet pour décrire et concevoir des systèmes orientés objet. Dans une dernière partie, nous détaillons plus particulièrement le diagramme de classes UML que les apprenants sont amenés à construire dans notre situation d'apprentissage et quelques représentations graphiques et internes.

## 1.1 La modélisation

Avant de discuter de la modélisation orientée objet et de son enseignement dans les cursus universitaires en informatique, il convient d'étudier les concepts et la terminologie associés aux notions de **modélisation** et de **modèle**. La modélisation peut être vue comme *« l'action de modéliser ou le résultat de cette action »* où **modéliser** consiste à *« concevoir, élaborer un modèle permettant de comprendre, d'agir, d'atteindre un but »* [Wiktionnaire]. Derrière cette définition très générale se cache en réalité une multitude d'interprétations et de terminologies.

Pour lever certaines ambiguïtés, nous exposons tout d'abord dans cette partie la terminologie associée aux modèles et à la modélisation dans le contexte de l'informatique. Nous présentons ensuite les différentes étapes de l'activité de modélisation. Enfin, nous définissons les notions de métamodèle et de langage intrinsèques aux modèles en informatique.

### *1.1.1 Des modèles à la modélisation*

La modélisation vise de manière générale à concevoir des modèles mais il n'existe pas à ce jour de définition universelle de ce qu'est un modèle. Toutefois, une étude plus approfondie de la littérature permet de mettre en exergue un relatif consensus sur une certaine compréhension de la notion de modèle.

Des chercheurs de la communauté IDM (**I**ngénierie **D**irigée par les **M**odèles) comme [Kühne 2005], [Kühne 2006] et [Favre *et al.* 2006] ont notamment utilisé le principe d'unification pour faire ressortir les caractéristiques d'un modèle et ont essayé de proposer une réponse à la question :

*Qu'est qu'un modèle ?*

Un modèle doit posséder trois caractéristiques [Stachowiak 1973] :

- Une **caractéristique d'application** (***mapping feature***) : un modèle est basé sur un original.
- Une **caractéristique de réduction** (***reduction feature***) : un modèle reflète seulement une sélection (pertinente) des propriétés de l'original.
- Une **caractéristique pragmatique** (***pragmatic feature***) : un modèle a besoin d'être utilisable à la place d'un original en respectant certains buts.

Les deux premières caractéristiques mettent en avant le fait qu'un modèle est une **projection théorique** impliquant que quelque chose est projeté (l'original) et que certaines informations sont perdues lors cette projection [Kühne 2005]. Un modèle est ainsi une **simplification** du monde, *« un filet jeté sur le monde qui permet d'attraper certains poissons mais en laisse échapper d'autres trop petits pour ses mailles »* [Magnin 2006].

L'original représenté par un modèle peut être construit ou exister dans la réalité (un avion, un chien, une carte topographique) mais également être imaginaire en se référant à des idées, des concepts abstraits tels que des notions de cours, de session, de transaction, de rôle ou de décision. Favre propose donc de qualifier avec le terme « système » l'original représenté par un modèle. Le terme n'a aucune importance en soi car l'idée sous-jacente est qu'il considère que *« tout est système »* et qu'il est pratique de les regrouper en trois grandes classes. Cette distinction n'est ni fondamentale, ni précise mais permet de montrer la généralité de son approche [Favre *et al.* 2006] : Les **systèmes physiques** sont concrets, observables et appartiennent au monde physique. Les **systèmes numériques** sont formés d'une séquence de bits, résident et sont manipulés par un ordinateur (c'est à ce genre de système que l'on s'intéresse en informatique). Enfin les **systèmes abstraits** sont immatériels, typiquement manipulés par le cerveau humain (un cercle, un compte bancaire, des entités mathématiques). L'auteur insiste sur le fait que cette classification peut être affinée en utilisant des dimensions plus ou moins orthogonales pour faire apparaître les notions de systèmes réactifs, temporels, événementiels, temps réels, hybrides, etc.

Un modèle, comme représentation formelle d'un système, **n'en est jamais une réplique exacte.** C'est plutôt une image stylisée et abstraite qui a comme but de représenter les aspects essentiels de la structure, des propriétés ou du comportement de ce dont il est le représentant [Politis *et al.* 2001]. Ainsi, un modèle est une **abstraction** qui permet de réduire la complexité en se focalisant sur certains aspects, en fonction de certains buts [Aussenac-Gilles & Charlet 2000]. Plus le problème est complexe et plus la nécessité de le modéliser se fait sentir [Crampes 2003].

Un modèle permet de mieux comprendre un problème complexe et également de communiquer des connaissances à autrui : un modèle est aussi une information sur quelque chose (un contenu, une signification) créé par quelqu'un (émetteur) pour quelqu'un d'autre (destinataire) dans un but donné [Steinmüller 1993]. Il peut être intéressant dans certains cas d'utiliser des modèles différents, correspondant à différents points de vue. De plus, la comparaison des résultats fournis par différents modèles est généralement très instructive sur les modèles et les phénomènes modélisés [Magnin 2006].

La modélisation, composant essentiel de l'activité humaine, se représente du moment où l'homme essaye de comprendre, d'interpréter les divers phénomènes du monde et de faire des prévisions [Politis *et al.* 2001]. Dans la vie

courante, nous modélisons tous et tout le temps : à chacun des êtres qui nous entourent, qu'il s'agisse d'objets matériels, de personnes ou d'institutions, nous associons une image mentale (modélisation implicite) qui nous permet d'anticiper leurs comportements [Volle 2004]. Cette démarche, intuitive et rapide dans la vie personnelle a pris aujourd'hui une part importante dans certaines activités professionnelles mais repose sur l'utilisation d'un vocabulaire technique, « abstrait » et éloigné du langage courant. Cette modélisation explicite est plus difficile à comprendre pour certaines personnes ; son enseignement s'est donc généralisé dans les cursus universitaires et plus particulièrement en informatique où l'on peut citer la modélisation de schémas relationnels pour les bases de données ou la modélisation de programmes orientés objet.

Dans le contexte de l'informatique, la modélisation est entendue comme étant la construction d'une **représentation schématique** (le modèle) dans un **langage formel ou semi-formel**, à partir de **spécifications données en langage naturel**.

### *1.1.2 L'activité de modélisation*

Selon [Komis *et al.* 2003], l'élaboration et l'utilisation de modèles font appel à des fonctions cognitives de formalisation, de conceptualisation et de raisonnement. D'un point de vue pédagogique, l'activité de modélisation peut renforcer le processus d'apprentissage puisque l'élève exprime ses idées et ses modèles mentaux dont il n'est pas forcément conscient : *« Un modèle, de par sa représentation graphique, permet aux idées abstraites de revêtir un aspect concret »*.

Une étude de l'activité de modélisation a été réalisée sur trois groupes d'élèves dans l'environnement ModelsCreator [Politis *et al.* 2001] conçu pour l'apprentissage de la résolution de problèmes dans des activités de modélisation scientifique par des élèves de collège. Dans [Komis *et al.* 2001], les auteurs ont pu constater que le processus de modélisation des élèves a été décomposé en une série de sous-tâches (des unités d'actions ayant un but spécifique) selon lesquelles les élèves ont procédé pour arriver à un modèle approprié. Cette décomposition en plusieurs tâches et sous-tâches de la modélisation d'un problème est reportée dans la figure 1.

1. Etude de la situation à modéliser
2. Objets du modèle
    - 2.1. Sélection des objets du modèle
    - 2.2. Placement des objets dans l'espace de travail
    - 2.3. Effacement des objets dans l'espace de travail
3. Propriétés des objets
    - 3.1. Sélection de la propriété de l'objet
    - 3.2. Etude de l'influence de la propriété sur le comportement d'objet
    - 3.3. Modification de la propriété
4. Relations
    - 4.1. Sélection des objets à lier
    - 4.2. Spécification de la relation
    - 4.3. Placement de la relation
    - 4.4. Modification de la relation
5. Vérification et modification
6. Vérification de la validité du modèle
7. Modification du modèle

**Figure 1 : Tâches et sous-tâches de l'activité de modélisation [Komis *et al.* 2001]**

Les tâches et sous-tâches ne sont pas accomplies de façon consécutive mais s'entrelacent pendant l'activité de modélisation. L'enchaînement des tâches dépend notamment des résultats des interactions de l'apprenant avec l'environnement et de la situation-problème à modéliser. Cette classification en tâches/sous-tâches est applicable à la modélisation d'un diagramme de classes UML en notant que la tâche 3.2 ne peut pas être réalisée en UML si l'on ne fait pas intervenir des modèles dynamiques.

### 1.1.3 Des modèles aux métamodèles et langages

Les modèles permettent de communiquer des connaissances entre différents acteurs. Pour que le sens d'un modèle (le *fond*) soit compris, le formalisme utilisé pour ce dernier (la *forme*) ne doit pas être un obstacle à la compréhension. Les acteurs ne peuvent en effet communiquer et se comprendre correctement que s'ils manipulent les bases d'un langage commun pour représenter leurs modèles. De plus, pour qu'un modèle soit **productif**, il doit pouvoir être manipulé par une machine. Le langage dans lequel ce modèle est exprimé doit être clairement défini pour les utilisateurs et la machine : l'IDM (**I**ngénierie **D**irigée par les **M**odèles) et la MOO (**M**odélisation **O**rientée **O**bjet) avec UML font explicitement référence à la notion de **langage bien-défini** et à la notion de **métamodèle** pour concevoir des modèles.

Contrairement aux idées parfois véhiculées, un *métamodèle n'est pas un modèle d'un modèle* [Favre *et al.* 2006]. Un métamodèle peut être défini ainsi :

> *« Un métamodèle est un modèle qui définit le langage d'expression d'un modèle. »* [OMG MOF]

> *« Un métamodèle est un modèle de spécification d'une classe de systèmes où chaque système de la classe est lui-même un modèle valide exprimé dans un certain langage de modélisation. »* [Kleppe *et al.* 2003]

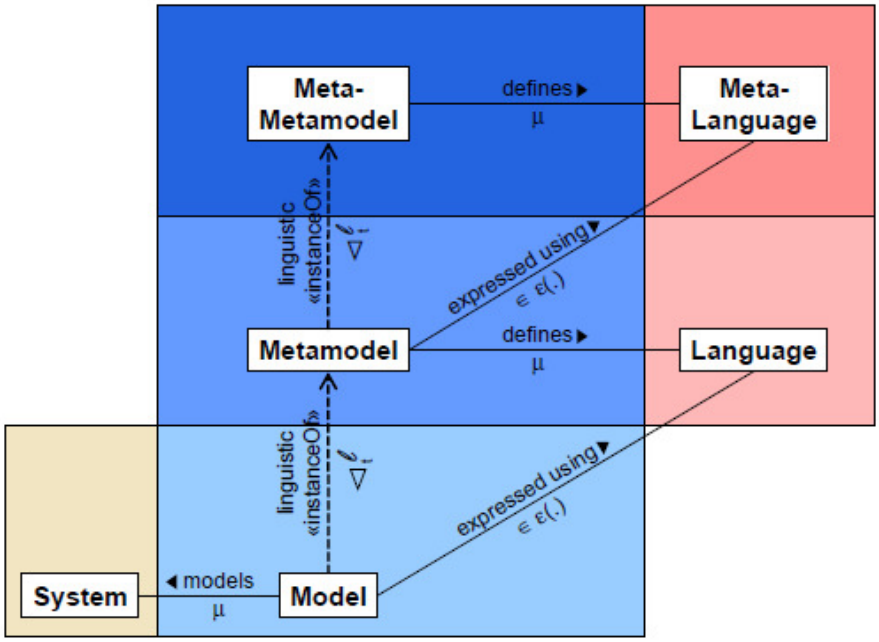



**Figure 2 : Métamodèles et définitions de langage [Kühne 2006]**

Un métamodèle est un moyen concret de définir un langage mais ce n'est pas langage [Favre *et al.* 2006]. C'est un modèle d'un langage de modélisation qui permet de limiter les ambiguïtés et de classer les différents concepts du

langage (selon leur niveau d'abstraction ou leur domaine d'application) et expose ainsi clairement sa structure [Piechoki]. Il est communément accepté dans la communauté IDM que la relation de conformance (*ConformeA*) lie un modèle à un métamodèle et que la relation de représentation (*RepresentationDe*) lie un modèle au système[1] qu'il modélise [Bezivin 2005]. Néanmoins, ces deux relations peuvent être vues comme des relations d'instanciation (*InstanciationDe*) comme c'est le cas dans le schéma de la figure 2 défini par Kühne.

## 1.2 La modélisation dans le paradigme *Objet*

En informatique, la modélisation est une activité cruciale intervenant essentiellement dans les étapes d'analyse et de conception du processus de développement d'un produit logiciel. De nombreux paradigmes en informatique (i.e. des manières d'analyser le monde dans le but de concevoir un programme informatique) existent et reposent sur différents cadres conceptuels. Ces paradigmes (de programmation) [Wikipedia] [Van Roy & Haridi 2004] peuvent être classés[2] ainsi :

- **le paradigme déclaratif** consiste à décrire le *quoi*, c'est-à-dire le problème. Les applications créées ne comportent aucun état interne. De nombreux paradigmes en découlent dont les suivants :
    - **le paradigme descriptif** permet notamment de décrire des structures de données.
    - **le paradigme logique** exprime les problèmes et les algorithmes sous forme de prédicats. Un programme logique est un ensemble d'axiomes (faits et règles de déduction).
    - **le paradigme fonctionnel** perçoit le système comme un ensemble de valeurs et de fonctions au sens mathématique du terme. Le calcul en tant qu'évaluation de fonctions mathématiques rejette le changement d'états et la mutation des données.
- **le paradigme impératif** consiste à décrire le *comment*, c'est-à-dire une solution : ce sont les opérations d'un système en termes de séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme (les changements d'états).
    - **le paradigme procédural** est basé sur le concept d'appel procédural et d'équations algorithmiques (des algorithmes). Une procédure contient simplement une série d'étapes à réaliser.
- **le paradigme multi-agent** est basé sur le fait que le monde est plein d'activités indépendantes et d'événements inattendus. Le système est modélisé en termes d'« agents » indépendants qui interagissent pour réaliser le but global du programme.
- **le paradigme orienté objet** consiste en la définition et l'assemblage de briques logicielles appelées objets (des instances de classes) qui communiquent entre eux, et travaillent ensemble.

Dans notre cas, nous intéressons uniquement à la modélisation dans le paradigme objet ou orienté objet qui dénote toute une philosophie du développement de systèmes qui englobe l'analyse des besoins, la conception de systèmes, la conception de bases de données, la programmation et toutes sortes de préoccupations voisines [Graham 1997]. La modélisation dans ce paradigme est nommée **modélisation orientée objet** ou **modélisation objet**.

[1] Un système peut être physique, numérique ou abstrait et aucun système ne peut être intrinsèquement un modèle. Néanmoins, un système peut jouer le rôle de modèle par rapport à un autre système.

[2] Cette classification simplifie les paradigmes existants en informatique et ne se veut pas complète. Van Roy dénombre notamment plus de 20 paradigmes différents et souligne qu'ils sont tous importants [Van Roy] [Van Roy & Haridi 2004]. Il précise qu'un langage qui ne soutient qu'un paradigme est un langage défectueux dans lequel il n'est pas possible de donner une bonne solution à tous les problèmes. C'est pour cela que la plupart les langages de programmation soutiennent des paradigmes multiples.

Elle consiste à créer une représentation informatique des éléments du monde auxquels on s'intéresse, sans se préoccuper de l'implantation (indépendamment d'un langage de programmation). Dans cette partie, nous exposons tout d'abord les principaux concepts du paradigme objet. Ensuite nous définissons brièvement les phases générales utilisées en génie logiciel lors du développement d'un logiciel et nous situons la part de modélisation dans chacune de ces phases. Nous nous attardons enfin sur les principales approches de l'enseignement de la modélisation orientée objet et sur leurs finalités dans les cursus universitaires.

### *1.2.1 Les concepts orientés objet*

Le paradigme orienté objet repose sur plusieurs concepts fondamentaux indépendants du langage de représentation utilisé lors de la construction d'un modèle objet : ces concepts sont les objets (et les classes d'objets), l'héritage, l'encapsulation, les messages, et le polymorphisme [Graham 1997].Nous ne présenterons ici que les concepts d'objet, de classe et d'héritage nécessaires à la compréhension de nos travaux.

Les **objets** sont des unités de base de construction, que ce soit pour la conceptualisation, l'analyse ou la programmation. Les objets reposent dans la mesure du possible sur des entités du monde réel et des concepts de l'application ou du domaine concerné [Graham 1997]. Une **classe** est une description d'un ensemble d'objets ayant des caractéristiques et un comportement communs : des attributs, des opérations (ou méthodes) et des relations (avec les autres objets du système). Un objet est une instance d'une classe et les différentes instances partagent un objectif sémantique commun qui dépend du point de vue de modélisation. Une classe est donc une définition et une **abstraction** des objets qu'elle représente : si l'on cherche « ordinateur » dans le dictionnaire, c'est sa définition que nous trouvons et non pas l'objet lui-même. Un attribut décrit l'apparence et la connaissance d'une classe d'objets, c'est une définition des données [Penders 2002] et donc de ce que les objets de cette classe ont la responsabilité de savoir. Une méthode (ou opération) définit le comportement qu'une classe d'objets peut manifester [Penders 2002] et donc de ce que les objets de cette classe ont la responsabilité de faire.

L'**héritage** est une des structures conceptuelles avec lesquelles nous organisons le monde. Elle est particulièrement importante car elle correspond au verbe *être* dans notre langue [Graham 1997]. Les objets héritent de tous les traits des classes auxquelles ils appartiennent, et uniquement de ceux-ci, mais un système à objets autorise également les classes à hériter des traits de superclasses plus générales. La notion d'héritage dans le paradigme objet est différente de celle en IA (**I**ntelligence **A**rtificielle) : les objets héritent des attributs et des méthodes de leurs classes ascendantes mais n'héritent pas des valeurs comme c'est le cas pour l'héritage en IA [Graham 1997]. L'héritage correspond à une relation « est un » ou « est une sorte de » entre les concepts spécialisés et les concepts généraux. Il permet d'éliminer les redondances de stockage des données ou des procédures et structure les classes en **hiérarchies « sorte de »**, ou **hiérarchies d'héritages**, ou **classifications**. Dans ces hiérarchies, la **généralisation** est l'abstraction des caractéristiques et des comportements communs à plusieurs classes d'objets alors que la **spécialisation** est la dérivation de nouvelles classes contenant des caractéristiques et des comportements supplémentaires [Graham 1997]. Par exemple, dans le métamodèle UML, les classes, les objets et les interfaces sont des sortes d'éléments nommés plus spécialisés (cf. figure 5 de la partie 1.3.3). Les classes « Classe », « Objet » et « Interface » sont des classes filles de la classe mère « Elément nommé ».

Un autre type de hiérarchies permet de structurer les objets autour de la notion d'agrégation ou de contenance. Ces hiérarchies sont nommées « partie de » (ou hiérarchies d'agrégation ou de composition). Par exemple, dans le métamodèle d'UML, une classe peut contenir des opérations qui elles-mêmes peuvent contenir des paramètres. Les objets issus de la classe « Paramètre » font partie des objets issus de la classe « Opération » qui font partie des instances de la classe « Classe ».

### *1.2.2 La modélisation dans les phases de développement d'un logiciel*

Les concepts objets que nous venons de décrire sont utilisés en génie logiciel pendant tout le cycle de développement d'une application. La modélisation tient une part prépondérante dans le **cycle de vie** d'un projet logiciel qui est constitué de l'ensemble des phases se déroulant depuis le moment où un client (ou un maître d'ouvrage) commande une application auprès d'un quelconque fournisseur jusqu'au moment où cette dernière est abandonnée par l'utilisateur. Les principales étapes du cycle de vie sont la spécification des besoins, l'analyse, la conception, la programmation, les tests, l'exploitation et la maintenance. La modélisation intervient essentiellement dans les phases d'analyse et de conception mais a des conséquences dans toute la suite du cycle de développement de l'application. Le cycle de vie d'une application logicielle peut être défini de manière générale comme dans la figure 3. Nous allons décrire brièvement en quoi consistent les différentes phases du cycle de vie d'un logiciel et la part de modélisation dans chacune d'entre elles.

Lors des **phases d'expression et de présentation des besoins**, le client et les informaticiens en charge du projet expriment de façon plus ou moins complète les besoins de l'application à réaliser. Un cahier des charges est notamment constitué sous forme d'un dossier structuré de spécifications en utilisant divers outils de représentation ou en les exprimant en langage naturel. Les premiers modèles généraux sont créés lors de cette phase.

La modélisation intervient pleinement à partir de la **phase d'analyse** car l'analyse réside notamment dans la construction d'un modèle du monde réel (**le modèle d'analyse**). Ce modèle met en avant les propriétés importantes du problème en une abstraction précise et concise du but de l'application et non de la façon dont elle sera bâtie. Les objets du modèle d'analyse sont des concepts du domaine d'application et non des objets informatiques tels que des tables, des fichiers, des fenêtres... [Graham 1997]. Les différents modèles créés lors de cette phase permettent de structurer les idées, de simuler le fonctionnement du réel, de communiquer à la fois entre informaticiens et avec le client pour « valider » les spécifications. La phase d'analyse introduit la phase de conception.

La **phase de conception** se focalise sur les prises de décisions de haut niveau concernant l'architecture de l'ensemble du système à mettre en œuvre. Elle fournit un support à l'implantation physique du système et traduit la solution dans une architecture réalisable informatiquement. Les modèles d'analyse servent de base aux modèles de conception qui sont plus détaillés et spécialisés. Ces modèles de conception traduisent ce que le code devrait être, les mécanismes et les composants réutilisables. La conception permet également de définir clairement les domaines techniques d'implantation tels que les langages de programmation, les systèmes d'exploitation, les gestionnaires de réseaux et de bases de données à utiliser.

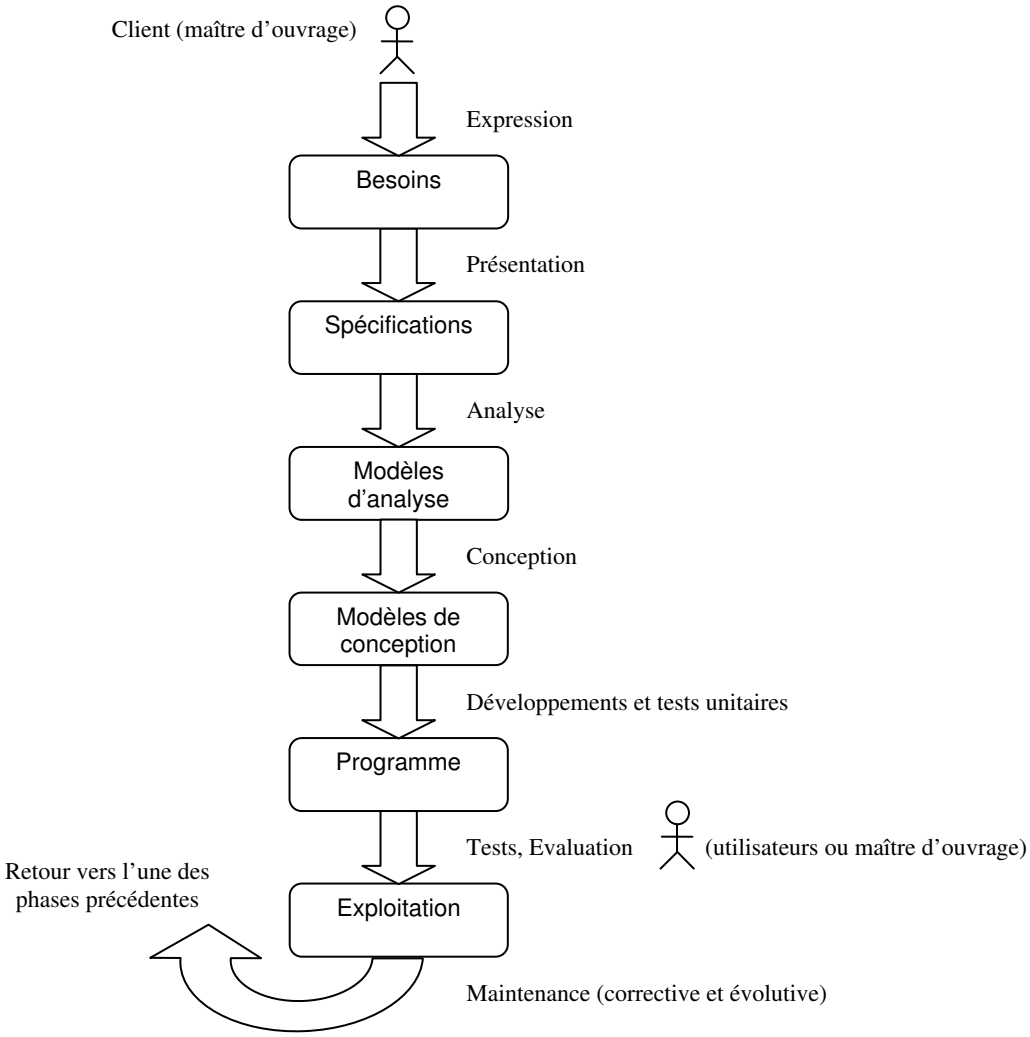



**Figure 3 : Cycle de vie d'une application [Crampes 2003]**

La **phase d'implantation** ou **de développement** concerne l'écriture des programmes (traitements, Interfaces Homme-Machine) dans des langages de programmation déterminés lors de la phase de conception, la saisie initiale des données contenues dans les fichiers ou les bases de données, ainsi que la mise en place des communications réseaux. La modélisation n'intervient pas directement dans cette phase mais a des conséquences sur la manière dont le système est programmé car (en simplifiant les choses) les modèles de conception compréhensibles par les informaticiens y sont « convertis » en code compréhensible et exécutable par la machine (cf. figure 4).

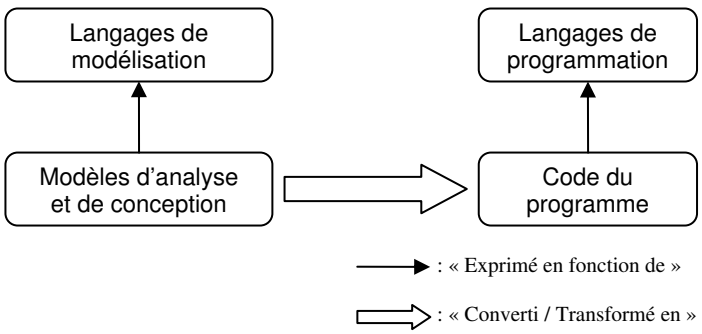



**Figure 4 : Simplification des activités d'analyse, de conception et de programmation**

Les **phases de validation et de tests** visent à valider d'une part le fonctionnement des traitements en vérifiant l'exactitude de tous les résultats produits en fonction des entrées, et d'autre part la logique de l'application, au travers de l'IHM (**I**nterface **H**omme-**M**achine) [Crampes 2003]. En fonction des anomalies constatées dans le programme (les bogues) et des remarques apportées par l'utilisateur, des corrections peuvent être apportées au programme. Ces modifications peuvent aller jusqu'à reprendre les spécifications des besoins dans les cas où l'application ou certaines fonctionnalités produites ne sont pas en adéquation avec les besoins réels de l'utilisateur. Les phases suivantes du cycle sont ensuite revues en cascade.

**La phase de maintenance** est semblable à la phase de validation mais elle est réalisée tout au long de l'exploitation du programme. Les modifications apportées à ce moment là peuvent être **correctives** (corrections de bogues résiduels) ou **évolutives** (par rapport à des modifications de l'application).

Dans nos travaux, nous nous intéressons plus particulièrement au début de la phase d'analyse, où l'activité de modélisation permet de définir à partir de spécifications textuelles les premiers modèles d'analyse qui constituent les bases du futur système à concevoir.

### *1.2.3 L'enseignement de la modélisation orientée objet*

L'acquisition des concepts et des bonnes pratiques orientés objet par les étudiants en informatique est importante car les erreurs relatives à l'analyse et à la conception sont toujours les plus coûteuses et les plus longues à corriger. En effet, ces phases orientent toute la logique de programmation des systèmes et des sous-systèmes de l'application. Une mauvaise maîtrise de la modélisation orientée objet peut être la cause d'erreurs se propageant dans tout le cycle de développement. Il faut donc acquérir de bonnes pratiques au plus tôt lors de l'apprentissage de la modélisation. Un enseignement adapté des concepts orientés objet lors de la modélisation, de la conception et de la programmation réduira fortement les surcoûts dans les projets logiciel auxquels participeront les futurs informaticiens. De plus, en milieu écologique les coûts sont minimes pendant les activités d'enseignement contrairement au milieu professionnel, où les coûts sont un facteur lourd sur le projet en lui-même.

L'enseignement des concepts orientés objet repose sur différents outils, approches et méthodes. Les méthodes adoptées ainsi que les difficultés d'apprentissage peuvent varier en fonction des points suivants :

- la nature du public cible ;
- les prérequis nécessaires et les connaissances déjà acquises ;
- les objectifs et les buts à atteindre ;
- les notions et les modèles précis à acquérir ;
- le ou les langages utilisés ;
- le positionnement dans le processus de développement d'un système orienté objet.

L'enseignement de la modélisation orientée objet peut s'adresser à la fois à des novices n'ayant pas ou très peu de connaissances sur le paradigme objet ou à des développeurs expérimentés. Une première question se pose donc :

*Peut-on aborder de la même manière l'enseignement de la modélisation orientée objet avec des novices et des développeurs expérimentés?*

Dans le cadre de l'analyse et de la conception de systèmes en génie logiciel, il est recommandé de centrer l'apprentissage dès le début sur les bonnes pratiques à acquérir permettant de développer une application orientée objet [Kuzniarz & Staron 2005]. Néanmoins, les novices demanderont une attention toute particulière, étant donné qu'ils ont tout à apprendre des concepts orientés objet et de la façon de les utiliser pour élaborer des modèles différents.

La modélisation orientée objet est souvent abordée après une première approche de la programmation orientée objet (avec un premier langage de programmation). Les apprenants ont ainsi des connaissances basiques sur les notions de classes, d'objets, d'encapsulation et d'héritage qu'ils perfectionnent lors de l'apprentissage de la MOO. Les apprenants peuvent avoir des difficultés dans ce cas à s'abstraire du ou des langages de programmation abordés et du futur code car ils ont le mauvais réflexe de penser aux détails de la « programmation » effective du système plutôt qu'à sa modélisation. Inversement, une stratégie incrémentale nommée *« Object-First »* [Moritz et *al.* 2005] répandue pour l'apprentissage de la programmation orientée objet favorise l'introduction des concepts d'objets, de classes et d'instances avant les éléments procéduraux d'un langage de programmation. Les étudiants apprennent donc ici à comprendre et à résoudre un problème, puis à concevoir une solution sans se soucier de la syntaxe des langages de programmation. L'étude de la programmation orientée objet n'est pas un prérequis nécessaire de la modélisation orientée objet. La MOO, la COO et la POO peuvent même être abordées conjointement au sein d'un même module d'enseignement ou parallèlement dans plusieurs.

La nature des problèmes à modéliser influe grandement sur l'apprentissage des concepts orientés objet. Nous pouvons différencier deux grandes façons de faire pour présenter les problèmes. Une première vision est d'utiliser des problèmes tirés de la vie réelle exprimés sous forme de spécifications plus ou moins détaillées telles qu'elles sont présentées dans un projet réel de développement. Le principal avantage de cette approche est que les problèmes à modéliser sont pertinents, c'est-à-dire similaires à des problèmes du monde réel qu'auront à traiter les étudiants quand ils seront en entreprise [Kuzniarz & Staron 2005]. Dans ce type d'approche, les apprenants peuvent être amenés à réaliser des projets, le plus souvent en groupe, pouvant aller de l'analyse à la programmation partielle ou complète d'un système orienté objet. Pour cela, ils construisent différents modèles qu'ils affinent et corrigent tout au long du cycle d'analyse et de conception de l'application. Les inconvénients majeurs sont que les spécifications sont souvent trop complexes et que les concepts sont difficiles à identifier dans ce genre de problème surtout pour des novices. De plus, de multiples représentations peuvent être produites pour une même spécification en fonction du point de vue du modélisateur. La seconde approche repose sur l'utilisation d'une série de problèmes simples à traiter. Chacun de ces problèmes se focalise sur des concepts et des difficultés particulières de la modélisation orientée objet. Les concepts sont beaucoup plus simples à identifier et l'espace de solutions est beaucoup moins grand. La plupart du temps, les exercices proposés sont de complexité croissante. Cette approche est la plus pertinente au début de l'apprentissage de la modélisation orientée objet afin que des novices acquièrent les bases des concepts orientés objet. Néanmoins, son inconvénient majeur est de masquer les difficultés réelles que les apprenants pourront rencontrer dans un vrai projet de développement de génie logiciel. Les problèmes sont souvent en effet trop simples, trop bien définis et relativement fermés. Pour préparer au mieux les étudiants à leur futur métier, il convient donc de focaliser l'enseignement tout d'abord sur des petits exercices et ensuite les impliquer dans des projets.

### 1.2.4 Les difficultés de l'activité de modélisation

La tâche de modélisation à un niveau abstrait peut être assimilée à une **tâche de conversion d'un registre (langagier) à un autre registre (symbolique)** nécessitant un changement fondamental des représentations elles-mêmes (il y a notamment un changement de systèmes de représentation). Cette tâche mobilise des connaissances et des savoir-faire spécifiques chez les apprenants. Pour pouvoir lire et comprendre les spécifications du problème de modélisation (énoncé textuel), l'apprenant doit notamment avoir des connaissances sur le langage naturel utilisé. Suraweera et Mitrovic expliquent également que la construction d'un schéma conceptuel (cela s'applique aux modèles orientés objet) nécessite d'avoir des connaissances sur le domaine concerné car les énoncés sont souvent longs, ambigus et incomplets [Suraweera & Mitrovic 2004]. D'autre part, pour modéliser certains éléments du modèle, l'élève doit réfléchir sur l'énoncé mais aussi sur ses propres connaissances. L'activité de modélisation n'est pas clairement définie et il n'existe donc pas d'algorithmes ou de stratégies prédéfinies pour réaliser un modèle à partir d'un énoncé donné. De plus, étant une simplification, tout modèle a ses limites, son domaine de validité : *« Le modèle n'est pas la réalité : en dehors de ses limites ses résultats ne collent plus à la réalité »* [Magnin 2006].

Plusieurs solutions pour un même énoncé sont possibles et elles dépendent aussi bien du formalisme, des niveaux d'abstraction, et de détails adoptés par l'apprenant, que de la perception qu'il a de l'énoncé en question. Le sens et l'interprétation d'un même modèle peut ainsi varier d'un lecteur à un autre au moment où il le lit et se l'approprie. Il est donc difficile dans ce contexte de définir clairement et de repérer les « erreurs » faites au cours d'une activité de modélisation du fait des différences d'appréciations de la réalité à modéliser de chaque personne.

Les apprenants peuvent être amenés à construire des modèles à différents niveaux d'abstraction car la modélisation prend part à la fois lors de l'analyse et de la conception orientée objet et elle a des conséquences directes sur la programmation orientée objet. L'utilisation de la syntaxe d'un langage de modélisation à différents niveaux d'abstraction est une source de confusion pour les étudiants. Par exemple, l'utilisation d'un diagramme de classes UML qui modélise le point de vue statique de l'architecture, de l'analyse et des modèles détaillés de conception représente un challenge pour les étudiants [Kuzniarz & Staron 2005].

L'apprentissage de la modélisation est une tâche longue et difficile que ce soit au niveau de la lecture, de la construction ou de la modification d'un modèle ; l'appropriation d'un modèle existant est complexe car elle demande de comprendre à la fois l'intention de l'auteur (le sens) et le formalisme utilisé (la syntaxe). La modélisation demande donc à l'apprenant de s'exercer sur de nombreux problèmes et exercices.

## 1.3 La modélisation orientée objet avec le langage UML

La modélisation a pris depuis ces cinq à dix dernières années une part importante des cursus informatiques universitaires à finalité professionnelle, surtout depuis l'avènement du langage de modélisation orienté objet UML. Les méthodes précédentes OOD (***O**bject-**O**riented **D**esign*) [Booch 1991], OMT (***O**bject-**M**odeling **T**echnique*) [Rumbaugh *et al.* 1991] et OOSE (***O**bject-**O**riented **S**oftware **E**ngineering*) [Jacobson *et al.* 1992] étaient très orientées vers les techniques de programmation et ont donné naissance après fusion à UML, une formalisation très aboutie et non-propriétaire de la modélisation orientée objet utilisée en génie logiciel.

UML [Booch *et al.* 1998] est une notation et un langage standardisé en 1997 par l'OMG (***O**bject **M**odeling **G**roup*) [OMG UML] qui facilite la conception de programmes, ainsi que leur description pour des non-informaticiens. UML a la particularité de s'intéresser essentiellement à la modélisation c'est-à-dire la représentation des différents concepts qui interviendront dans l'écriture d'un logiciel [Crampes 2003]. Il est devenu le langage de modélisation orienté objet de référence dans le monde professionnel car il comble une lacune importante des technologies objet : il permet d'exprimer et d'élaborer des modèles objet, indépendamment de tout langage de programmation [Piechoki]. Il a été pensé pour servir de support à une analyse basée sur les concepts objet. C'est également un langage graphique semi-formel orienté objet issu des meilleurs outils et pratiques du génie logiciel du début des années 90.

### *1.3.1 Formalisme d'UML*

Qu'un héritage entre deux classes soit représenté par une flèche terminée par un triangle ou un cercle n'a que peu d'importance par rapport au **sens** que cela donne au modèle. La notation graphique est essentiellement guidée par des considérations esthétiques, même si elle a été pensée dans ses moindres détails. Toutefois, utiliser une relation d'héritage reflète l'intention de donner à un modèle un sens particulier. Un « bon » langage de modélisation doit permettre à n'importe qui de déchiffrer cette intention de manière non équivoque. Il est donc primordial de s'accorder sur la sémantique des éléments de modélisation, bien avant de s'intéresser à la manière de les représenter. La notation graphique d'UML n'est que le support du langage et c'est pourquoi la véritable force d'UML repose sur son métamodèle. En d'autres termes : la puissance et l'intérêt d'UML sont qu'il **normalise la sémantique des concepts qu'il véhicule** [Piechoki].

Le métamodèle d'UML décrit de manière très précise tous les éléments de modélisation (les concepts véhiculés et manipulés par le langage) et la sémantique de ces éléments (leur définition et le sens de leur utilisation). Il permet aussi de classer les différents concepts du langage (selon leur niveau d'abstraction ou leur domaine d'application) et expose ainsi clairement sa structure [Piechoki]. On peut noter que le métamodèle d'UML est lui-même décrit par un méta-métamodèle de manière standardisée, à l'aide de MOF *(**M**eta **O**bject **F**acility*) [OMG MOF], une norme OMG de description des métamodèles.

À chaque nouvelle version d'UML, l'OMG définit ainsi une série de quatre documents exposant précisément la spécification complète d'UML et de sa sémantique :

- La **superstructure d'UML** définit les constructions de niveau utilisateur du métamodèle.
- L'**infrastructure d'UML** contient les classes de base formant non seulement les fondations de l'architecture de la superstructure d'UML mais aussi les bases de celles de MOF[3].
- Le **langage OCL** (***O**bject **C**onstraint **L**anguage*) fournit les propriétés des pré-conditions et post-conditions, des invariants, et d'autres conditions.
- « **L'échangeur de diagrammes** » est une spécification qui étend le métamodèle d'UML avec des informations orientées-graphe permettant à des modèles d'être échangés ou sauvegardés et ensuite affichés comme ils étaient originellement.

Il faut bien noter qu'UML est avant tout un langage et qu'à ce titre, il ne fournit pas un environnement d'interrogation ou de vérification de diagrammes. Or, la démarche de construction d'une modélisation nécessite de la part du concepteur de consulter des parties de diagrammes déjà écrites pour avancer. De plus, pour valider ses diagrammes, le concepteur devrait pouvoir préciser lui-même des spécifications sur ses diagrammes en vue d'une vérification automatique.

Malgré les avancées des dernières versions d'UML, ce langage n'a pas encore atteint la rigueur syntaxique et sémantique des langages de programmation, notamment au niveau de l'expression des contraintes et de la cohérence intra et inter-diagrammes.

### *1.3.2 Diagrammes d'UML*

La version actuelle du langage est UML 2.1. Elle définit treize types de diagrammes, répartis en trois catégories (cf. annexe 1) :

- Les **diagrammes de structure** incluent le diagramme de classes, le diagramme d'objets, le diagramme de composants, le diagramme de structure composite, le diagramme de paquetages, et le diagramme de déploiement.
- Les **diagrammes de comportement** incluent le diagramme de cas d'utilisation, le diagramme d'activités, et le diagramme d'états-transitions.
- Les **diagrammes d'interaction** (dérivés de la catégorie des diagrammes de comportement) incluent le diagramme de séquences, le diagramme de communication, le diagramme de synchronisation (ou de temps), et le diagramme de vue d'ensemble des interactions.

[3] MOF de l'OMG est la base des environnements standards de l'industrie où les modèles peuvent être exportés d'une application, importés dans une autre, transportés sur un réseau, sauvegardés dans un dépôt et ensuite retrouvés, et rendus dans différents formats (dont XMI un format standard de l'OMG basé sur XML pour la transmission et la sauvegarde), transformés, et utilisés pour générer du code.

Les diagrammes UML peuvent être utilisés à tous les niveaux de l'OOAD (***O**bject-**O**riented **A**nalysis and **D**esign*). Leur utilisation est subtilement différente à chaque niveau [Fowler 2003] :

Au **niveau de l'analyse et de la spécification des besoins** (*requierements analysis*) les diagrammes UML permettent de comprendre le domaine fondamental ; à ce moment, il n'y a pas besoin de penser aux relations avec le logiciel résultant. Les modèles du domaine sont des modèles conceptuels et ont une place importante pour débuter ensuite la conception. Les diagrammes de cas d'utilisation sont très souvent utilisés à ce moment pour capturer les besoins des utilisateurs.

Au **niveau de l'analyse** (*analysis*) et toujours dans une optique d'indépendance avec l'implantation, l'objectif est de modéliser le monde réel en déterminant les classes d'objets du monde réel dans un premier diagramme de classes et de modéliser également la dynamique du système notamment par un diagramme de collaboration. Les diagrammes d'activités et d'états-transitions peuvent être également utilisés.

Au **niveau de la conception** (*design*), l'architecture informatique est prise en compte et les entités des diagrammes sont examinées comme des spécifications de ce que le code devrait être. C'est souvent dans cette perspective que les diagrammes UML sont les plus utilisés. À ce niveau, des classes techniques sont ajoutées pour gérer l'interface graphique, la distribution, la persistance et la concurrence. Les diagrammes manipulés lors de cette phase sont les diagrammes de classes, de séquences, de composants, de déploiement et d'états-transitions.

Au **niveau de l'implantation** (*programming*), les classes de conception sont converties vers les langages cibles (par exemple Java, SQL, C++, IDL) et les classes persistantes sont converties vers des modèles de persistances (par exemple SGBD, BDOO, langages persistants). L'attention portée sur les diagrammes à ce niveau est concentrée sur les attributs et les opérations principales pour que « l'image » du système ne soit pas trop grande.

Dans la **phase de tests**, les diagrammes UML définis dans les phases précédentes permettent de diriger des tests unitaires, des tests d'intégration et des tests du système codé de manière générale. Les diagrammes de classes sont utilisés notamment pour mener à bien des tests unitaires classe par classe et méthode par méthode alors que les diagrammes de composants sont plus utiles pour les tests d'intégration. Les diagrammes de cas d'utilisation et d'activités permettent enfin de vérifier la conformité des fonctionnalités implantées dans le système.

UML propose plusieurs sortes de diagrammes, dont le diagramme de classes qui nous intéresse plus particulièrement. Le diagramme de classes est le plus employé et le mieux connu des diagrammes orientés objet ; il est considéré comme le plus important des diagrammes du fait qu'il est le cœur de la modélisation objet. Piechocki définit un diagramme de classes comme une collection d'éléments de modélisation statique qui fait abstraction des aspects dynamiques et temporels [Piechocki]. Un diagramme de classes est constitué essentiellement de classes liées entre elles par des relations. Les diagrammes de classes sont utilisés à la fois dans les phases d'analyse et de conception et peuvent représenter les informations avec différents niveaux d'abstraction et de précision en fonction de leur utilisation et de leur degré d'affinement.

### *1.3.3 Diagramme de classes UML*

Les différents éléments représentables dans un diagramme de classes UML sont décrits par le métamodèle d'UML défini par l'OMG dans le document de spécification de la superstructure d'UML. La figure 5 résume sous forme hiérarchique (un diagramme de classes UML traduit en français) un extrait simplifié du métamodèle UML focalisé uniquement sur les éléments représentables dans les diagrammes de classes UML. Une version plus complète du métamodèle originel est reportée dans l'annexe 2.

Le langage UML 2.x organise les éléments des diagrammes de classes en trois grandes catégories :

- **Les classificateurs (*classifiers*)** ont des caractéristiques communes (des propriétés et/ou des opérations) et sont qualifiés par un espace de nommage unique (un nom). Les classificateurs sont généralisables et peuvent être liés entre eux par des relations.
- **Les relations (*relationships*)** sont les connexions logiques entre les classificateurs apparaissant dans un diagramme. Elles peuvent être dirigées (héritage, dépendance) ou non (association). Les relations ont une structure et un sens bien définis en fonction de leur type.
- **Les caractéristiques (*features*)** sont encapsulées dans les classificateurs. Elles définissent leurs propriétés (attributs) et leurs comportements (les méthodes) communs.

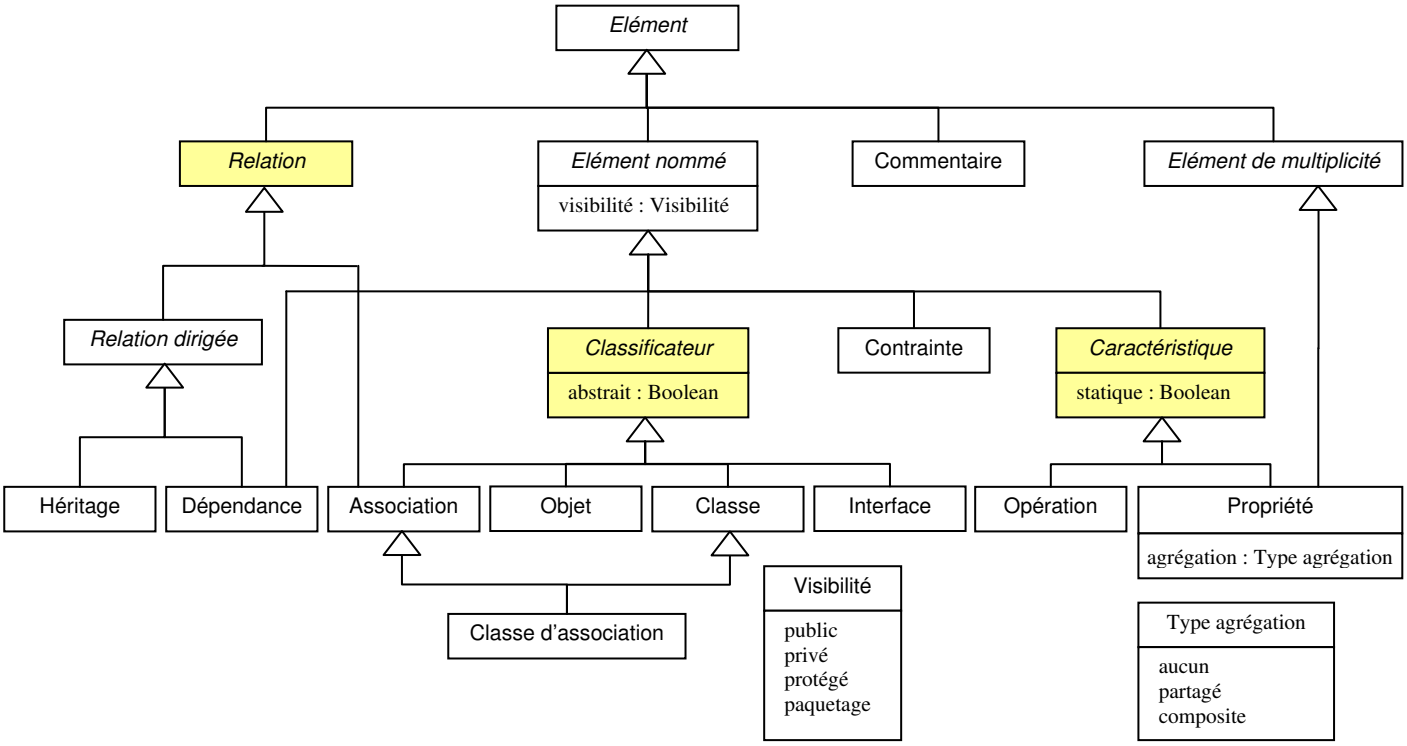



**Figure 5 : Extrait simplifié du métamodèle pour les diagrammes de classes UML 2.x**

#### 1.3.3.1 Les classificateurs

Les classificateurs les plus employés dans les diagrammes de classes sont les **classes**. Une classe est représentée graphiquement sous forme d'un rectangle découpé en trois parties : le nom de la classe, les attributs et les

méthodes encapsulées dans cette dernière. Les noms des classes portent par convention une majuscule en tête. Dans l'exemple suivant, la classe représentée est nommée « Personne » et possède trois attributs (« nom », « date de naissance » et « age ») et une méthode qui permet d'accéder à la valeur de l'attribut « age ». Pour ne pas surcharger les diagrammes de classes, les objets sont habituellement représentés à part dans des diagrammes d'objets permettant de mettre en valeur les objets instanciés et leurs liens effectifs.

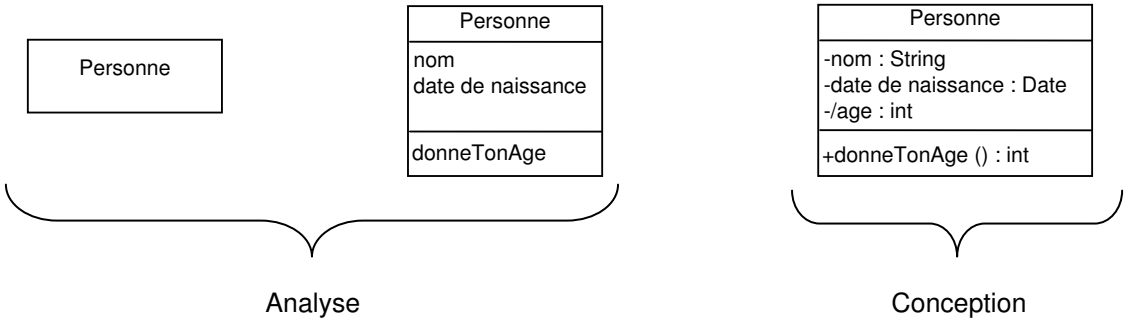



**Figure 6 : Représentations graphiques plus ou moins détaillées d'une classe en UML**

Une classe peut être représentée avec plus ou moins de « détails » si elle apparaît dans un diagramme de niveau analyse ou de niveau conception. Dans un diagramme d'analyse, une classe peut être représentée par son nom uniquement (ses propriétés et son comportement sont masqués), ou bien par son nom, les noms de ses attributs et (au besoin) de ses opérations encapsulés. Au niveau de la conception, il est obligatoire cependant de représenter les classes de manière plus complète en indiquant notamment les types (pouvant être précisés également en analyse), les valeurs par défaut des attributs et les signatures complètes des opérations[4] (avec leurs paramètres d'entrée et de sortie nommés et typés). La visibilité (publique, protégée, privée, paquetage) des attributs et des opérations doit être indiquée au moment de la conception car elle est très importante ensuite lors de l'implantation des classes dans le programme. Par exemple, un attribut privé ne sera visible et utilisable que par les instances de la classe le contenant alors qu'un attribut protégé pourra être utilisable et visible en plus par les classes filles de cette classe.

Un type particulier de classe est la **classe abstraite**. Elle a une représentation similaire à celle d'une classe (concrète) mais ne peut être instanciée directement sous forme d'objets. Pour différencier graphiquement une classe abstraite d'une classe (qui ne l'est pas), son nom est écrit en italique ou précédé d'un stéréotype particulier. Les classes abstraites permettent de généraliser des propriétés et un comportement commun à plusieurs autres classes plus spécialisées (reliées par des relations d'héritage). Les classes abstraites sont utilisées dès la phase d'analyse du problème et prennent tout leur sens au niveau de la phase de conception.

Une **classe interface** ou **interface** est aussi un classificateur qui ne peut pas servir à instancier des objets. Elle définit seulement les signatures de ses opérations. Ce type fournit une vue totale ou partielle d'un ensemble de services offerts par une classe [Piechocki]. Les classes qui implantent une interface (avec une relation de dépendance de réalisation) peuvent exploiter tout ou partie de l'interface. Les interfaces sont généralement utilisées dans les diagrammes de classes de niveau conception pour introduire des structures particulières de programmation et sont une manière de contourner les problèmes induits par l'héritage multiple.

[4] Une opération est une définition d'un service qui peut être demandé à une instance de la classe. Une méthode est une implantation d'une opération dont elle spécifie l'algorithme ou la procédure associée. On ne parle donc que d'opérations au niveau de l'analyse et de méthodes dans les diagrammes de classes les plus affinés de niveau conception préparant la phase de programmation.

#### 1.3.3.2 Les relations

Les relations liant les classificateurs peuvent être de natures diverses. Une **association** est une relation sémantique entre des classes qui définissent un ensemble de liens [Piechocki]. L'association est représentée sous forme d'une ligne entre les classes participantes. Le but d'une association peut être exprimé par un verbe ou un syntagme verbal décrivant comment les objets d'une classe sont en relation avec ceux d'une autre classe [Penders 2002]. Les associations entre plus de deux classes (**associations n-aires**) sont peu utilisées et sont rarement recommandées du fait de leur difficulté de déchiffrage et des erreurs qu'elles peuvent induire. Chaque extrémité d'une association (fin d'association) est une propriété dans le métamodèle d'UML (tout comme les attributs dans les classes). Une fin d'association définit le rôle de la classe liée (optionnel) ainsi qu'une multiplicité. Les multiplicités précisent combien d'objets des différentes classes interviennent dans la relation. Une multiplicité doit être affectée à chacune des classes participant à une association et doit suivre la syntaxe définie dans le tableau de la figure 7.

| Symbole | Signification |
|---|---|
| 1 | une seule instance. |
| 0..1 | de zéro à 1 instance. |
| M..N | de M à N instances. M < N, M et N sont des entiers non négatifs. |
| * | de 0 à un nombre quelconque d'instances. |
| 0..* | identique à *. |
| 1..* | de 1 à un nombre quelconque d'instances. |
| M..* | de M à un nombre quelconque d'instances. M est un entier non négatif. |

**Figure 7 : Multiplicités syntaxiquement correctes pour les propriétés en UML**

Les **agrégations** et les **compositions** sont des associations binaires particulières et plus restrictives qui apportent le sens « est une partie de » à la relation. L'agrégation indique que les objets sont plus que des objets indépendants ayant des connaissances les uns des autres : ces objets (les composants) peuvent être **assemblés** ou **configurés** ensemble pour créer de nouveaux objets plus complexes (les agrégats). Un losange est reporté à l'extrémité de la relation du côté de l'agrégat pour différencier graphiquement une agrégation d'une simple association. La composition est une agrégation où la durée de vie des participants dépend de celle de la relation les reliant. Les objets (les composants) entrant dans la composition d'autres objets (les composites) ne peuvent exister seuls en dehors. Cette forme plus forte d'agrégation est indiquée graphiquement à l'aide d'un losange noir. La figure suivante illustre les relations d'inclusion existant entre les différents types d'association que nous venons de définir :

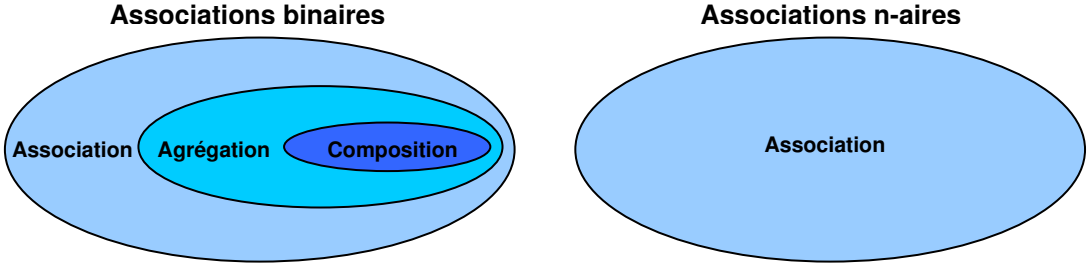



**Figure 8 : Différents types de relations d'association dans les diagrammes de classes**

L'exemple suivant montre les différences entre les trois types d'associations binaires que nous avons explicitées précédemment. La première partie insiste sur le fait qu'il peut y avoir plusieurs relations d'association distinctes entre deux mêmes classes et qu'elles expriment des buts différents : le fait qu'une personne puisse conduire une voiture (ou pas) n'a pas de rapport direct avec le fait qu'une personne (le propriétaire) possède plusieurs voitures. Il est donc nécessaire de représenter ces relations distinctement. La différence entre une agrégation et une composition ressort dans le reste de l'exemple : elle réside essentiellement dans le fait que lorsqu'une équipe de neuf joueurs est dissoute, les membres continuent d'avoir une existence propre. De plus, les joueurs existent également avant que l'équipe ne soit créée. En revanche, la création d'un immeuble entraîne la création des appartements le constituant et la destruction de l'immeuble entraîne celle des appartements également.

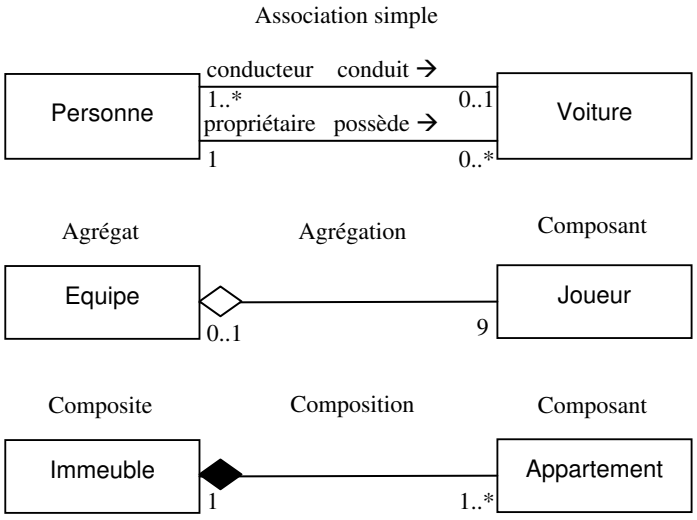



**Figure 9 : Différents types d'association binaire en UML**

Un dernier type d'association parfois difficile à comprendre existe en UML : c'est la **classe d'association**. Une classe d'association possède à la fois les comportements d'une classe et d'une relation d'association. Elle encapsule les informations concernant l'association dans une classe et elle est utilisée pour modéliser une association qui possède des caractéristiques propres extérieures aux classes liées. Dans la partie droite de la figure 10, la classe « C », la relation entre les classes « A » et « B » ainsi que le lien en pointillé constituent une classe d'association. Cette représentation graphique est équivalente à la représentation de gauche : une classe d'association est dans ce cas un « raccourci » pour représenter sous forme d'une seule relation « plusieurs à plusieurs »[5] un ensemble de relations « un à plusieurs » entre des classes.

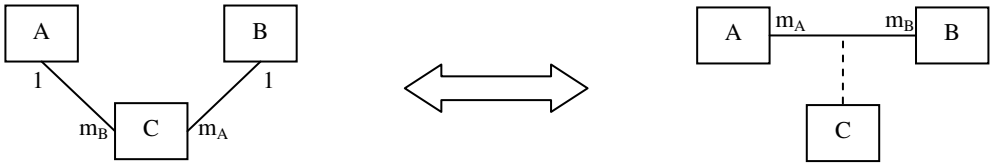



**Figure 10 : Représentations d'une classe d'association en UML**

Une autre forme de relation peut exister entre deux classes : le lien d'**héritage**. Cette relation est utilisée de manière courante pour organiser de grandes quantités d'informations [Penders 2002]. L'héritage est une relation transitive de classification qui permet de transmettre des caractéristiques et un comportement à ses descendants ; elle

[5] La relation « plusieurs à plusieurs » renvoie aux multiplicités $m_A$ et $m_B$ ayant pour valeurs « n ..* » avec $n \geq 0$.

correspond à une relation *« est un »* ou *« est une sorte de »*. La classe qui hérite (classe fille, descendante ou sous classe) dispose des attributs et des méthodes de niveau public et protégé de la classe mère. Une classe fille hérite également des relations liant ses classes mères à d'autres classes du modèle. Ces relations sont implicites et ne sont pas représentées pour la classe fille : dans le cas, où elles sont représentées, elles doivent apporter une information supplémentaire pour ne pas être redondantes : ces associations sont des associations dérivées notées avec un / devant leur nom. La relation d'héritage est notée graphiquement en UML sous forme d'un trait avec un triangle à l'extrémité correspondant à la superclasse (classe mère). Le métamodèle des éléments représentables dans un diagramme de classes (cf. figure 5) est un exemple de classification reposant sur l'héritage : tous les éléments d'un diagramme de classes héritent de la classe abstraite « Élément » qui est la racine de cette hiérarchie.

#### 1.3.3.3 La dérivation

Certains éléments d'un diagramme de classe peuvent être dépendants d'autres éléments d'un diagramme : ce procédé se nomme la **dérivation**. Un élément dérivable d'un autre élément entretient une relation de dépendance calculable avec ce dernier. Un élément dérivé est noté avec un / devant son nom. Les éléments dérivés représentés dans un modèle peuvent être considérés comme redondants étant donné qu'ils peuvent être calculés à partir d'autres éléments du modèle. De manière générale dans un diagramme de classes de niveau analyse, il n'est pas nécessaire de représenter les éléments dérivés. Les propriétés (attributs) et les associations peuvent être dérivées. Par exemple, dans la figure 6, l'attribut « age » est calculable à partir de l'attribut « date de naissance » et n'est représenté que dans le diagramme de classes de niveau conception pour éviter des calculs intermédiaires de l'âge.

## 1.4 Exemples de représentations du diagramme de classes UML

Nous avons détaillé les différents éléments constituant un diagramme de classes UML et situé leur usage au niveau des phases d'analyse et de conception. Dans notre contexte d'apprentissage, les modèles élaborés par les apprenants lors d'une activité de modélisation sont des diagrammes de classes UML graphiques de niveau analyse. Dans cette partie, nous étudions quelques unes des représentations possibles du diagramme de classes UML. Pour cela, nous expliquons tout d'abord la résolution d'un exercice de modélisation proposé aux novices dans Diagram et proposons une représentation graphique correcte de cet exercice. Ensuite, nous nous concentrons sur la manière dont les diagrammes de classes UML peuvent être représentés sous forme de graphes notamment pour être sauvegardés et analysés par des systèmes informatiques. Nous nous servons du même exercice pour illustrer les différentes représentations.

### *1.4.1 Exemple d'exercice de modélisation d'un diagramme de classes UML*

Nous nous focalisons dans nos travaux sur l'apprentissage des concepts orientés objet de niveau analyse par des étudiants novices. Nous allons donc maintenant illustrer l'emploi de ces concepts lors d'une activité de construction d'un diagramme de classes UML de niveau analyse à partir d'un énoncé textuel. L'exercice nommé « Stylo et Feutre » que nous présentons ici est relativement simple car son énoncé jouant le rôle de spécifications comporte seulement cinq phrases. Il permet d'apprendre les bases des concepts orientés employés dans les diagrammes de classes et est utilisé

actuellement au cours de séances de travaux pratiques avec l'environnement Diagram (cf. partie 2.4). Un exemple de diagramme de classes modélisant l'énoncé de l'exercice suivant est reporté dans la figure 11.

> *« Un **stylo** et un **feutre** sont deux concepts proches ayant des caractéristiques communes : couleur, marque, etc. Un **feutre** possède un **bouchon**. Un **stylo** et un **feutre** possèdent tous les deux un **corps** ayant certaines propriétés. Un **stylo** ou un **feutre** sont utilisés par une **personne** et appartiennent à une **personne**. Il existe un feutre particulier qui est un **feutre effaceur**. »*

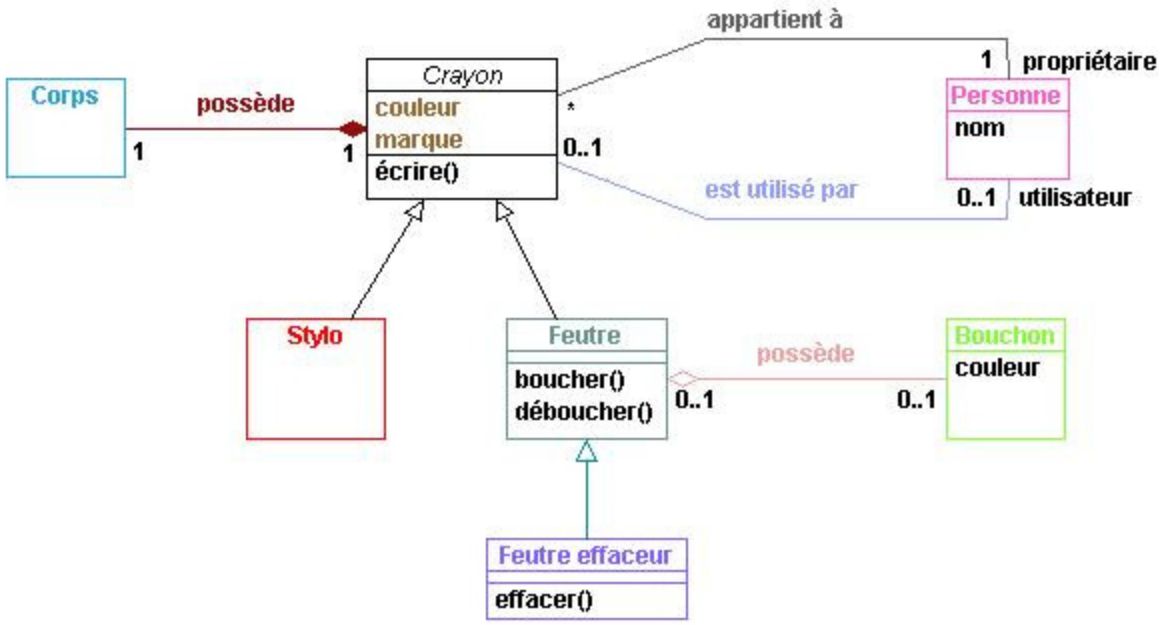



**Figure 11 : Exemple de diagramme de classes UML d'analyse pour l'exercice « Stylo et Feutre »**

Dans cet énoncé court, les classes du diagramme sont pratiquement toutes identifiables directement (elles sont reportées en gras dans l'énoncé) car elles sont représentées par des noms communs non ambigus et facilement différenciables. Malgré son apparente simplicité, cet exercice recèle des difficultés propres à l'apprentissage de la modélisation orientée objet telles que l'identification d'une classe implicite et la factorisation de certaines propriétés et de comportements dans une classe plus générale d'une hiérarchie de classes. Il ressort de l'énoncé que les propriétés « couleur » et « marque » et que certaines relations reliant les concepts « Feutre » et « Stylo » sont communs (cf. phrase 1 de l'énoncé). Il convient donc de les factoriser dans une classe plus générale que nous nommons « Crayon »[6] dans le diagramme de la figure 11. Cette classe est implicite dans l'énoncé mais elle est nécessaire pour ne pas avoir de redondances dans le diagramme. Les classes « Feutre » et « Stylo » héritent directement de la classe « Crayon » et possèdent ses propriétés et son comportement par héritage (propagation des propriétés et du comportement). Un type particulier de « Feutre » est défini dans l'énoncé, ce qui signifie qu'il faut représenter une classe « Feutre Effaceur » liée à la classe « Feutre » par un héritage (cf. phrase 5 de l'énoncé). Un feutre effaceur est ainsi une spécialisation d'un feutre et a comme comportement supplémentaire de pouvoir effacer du texte (une opération implicite dans l'énoncé).

Les classes « Crayon », « Feutre », « Stylo » et « Feutre effaceur » constituent l'unique hiérarchie d'héritages à modéliser dans ce diagramme de classes. La classe « Crayon » est abstraite car on ne peut créer ici que des objets de type « Feutre », « Stylo » ou « Feutre effaceur » définis dans l'énoncé. Elle constitue la racine de cette hiérarchie

[6] Le nom de cette classe n'est pas important. Elle pourrait très bien être appelée autrement, par exemple « Outil dactylographique ».

arborescente dont tous les types de crayons dérivent : des feutres, des stylos et des feutres effaceurs sont des sortes de crayon plus spécialisés.

Cet exercice permet aussi d'appréhender les différences qui existent entre les relations d'agrégation et de composition. Deux relations nommées « possède » (relation « est une partie de ») sont définies dans l'énoncé mais elles n'ont pas la même traduction dans le diagramme de classes. Il est précisé qu'un feutre peut posséder un bouchon (ou non), or un bouchon peut exister indépendamment du feutre qu'il peut boucher, donc un feutre est lié à un bouchon par une relation d'agrégation (cf. phrase 2 de l'énoncé). La seconde relation présente une double difficulté : un stylo ou un feutre possèdent un corps dont ils sont indissociables ; les objets de ces deux classes ont cette relation commune avec les objets de type « Corps ». Une relation unique de composition est donc reportée entre la classe « Crayon » et la classe « Corps » (cf. phrase 3 de l'énoncé). Les classes « Stylo », « Feutre » et « Feutre Effaceur » possèdent implicitement cette relation avec la classe « Corps » par héritage.

Cet exercice introduit une autre particularité difficile à appréhender par les étudiants au début de leur apprentissage : deux relations distinctes entre les deux mêmes classes. Ces relations sont des associations nommées « appartient à » et « est utilisé par » qui relient les objets héritant du type « Crayon » aux objets de type « Personne » (cf. phrase 4 de l'énoncé). En effet, l'énoncé distingue précisément les deux relations que doivent entretenir ces objets. Le fait qu'un crayon appartienne à une personne et qu'il soit utilisé par cette même personne ou une autre sont des concepts indépendants. Les rôles des objets de type « Personne » peuvent être précisés au niveau de chaque extrémité des associations du côté de la classe « Personne » (respectivement « propriétaire » et « utilisateur »).

Certains éléments de l'énoncé ne sont pas définis précisément, pour que le modélisateur puisse y réfléchir seul et représenter ce qui pourrait être nécessaire dans ce diagramme : « etc. » à la phrase 1 et « ayant certaines propriétés » à la phrase 3. En partant de ce principe et pour donner une vision un peu plus complète du futur système et de son comportement, nous avons représenté (en noir dans le diagramme) des attributs et des opérations qui ne sont pas explicitement définis dans l'énoncé : les crayons servent notamment à écrire par le biais d'une opération « écrire() ». Les bouchons des feutres peuvent être détachés et attachés par le biais de deux opérations distinctes « déboucher() » et « boucher() ». Un feutre effaceur peut effacer du texte écrit grâce l'opération « effacer() ». Un bouchon peut avoir une couleur différente de celle du crayon auquel il peut être rattaché grâce à l'attribut « couleur ». Enfin, une personne peut avoir un nom qui lui est propre contenu dans l'attribut « nom ».

### *1.4.2 Représentation du diagramme de classes UML sous forme de graphes*

La représentation graphique des diagrammes de classes UML, manipulée à l'interface par l'utilisateur, est rarement celle utilisée en interne par les outils de modélisation pour les représenter, les sauvegarder et les analyser. Dans cette partie, nous étudions trois représentations du diagramme de classes UML sous forme de graphes[7] pouvant soutenir l'analyse (notamment par des systèmes de vérification de la cohérence ou de la syntaxe des modèles ou encore des systèmes d'appariement) et la persistance des données véhiculées. La première découle directement de la

[7] Nous centrons notre étude sur les graphes car la majorité des systèmes d'appariement de modèles encodent les modèles à analyser sous forme de graphes (cf. chapitre 4). De plus, intuitivement les diagrammes de classes UML sont des graphes à l'échelle des classes et des relations graphiques.

représentation graphique des modèles des éditeurs UML classiques. La seconde est issue de la représentation interne de ces derniers sous forme d'arbre notamment dans le format XMI (**X**ML **M**etadata **I**nterchange) [OMG XMI]. La dernière est une variation de la représentation arborescente faisant ressortir tous les liens entretenus entre les différents éléments UML. Pour illustrer chacune des représentations, nous utiliserons comme exemple le diagramme de classes UML défini pour l'exercice « Stylo et Feutre » (cf. figure 11).

#### 1.4.2.1 Représentation graphique

Intuitivement, la représentation graphique des diagrammes de classes UML créés dans les éditeurs UML classiques est un **graphe orienté multiple** (où les arcs sont orientés et multiples entre les sommets et où les boucles et les cycles sont autorisés). Les sommets sont les classificateurs (classe, interface, objet, acteur...) et les arcs sont les relations (association, héritage, dépendance) reliant les classificateurs. Les nœuds et les arcs possèdent de nombreuses caractéristiques (des étiquettes) : les espaces de nommage, l'abstraction, la visibilité, les types de relations, les attributs, les opérations, les fins d'association, les types d'agrégation, etc. Cette vision revient à considérer un diagramme de classes dans son expression la plus simple, c'est-à-dire un ensemble de boites reliées par des traits. La figure 12 est un exemple de représentation du diagramme de classes détaillé pris en référence pour l'exercice « Stylo et Feutre » sous forme d'un graphe. Les sommets (les classes) sont des rectangles et les arcs sont des flèches (le type de chaque relation UML est reportée sur chaque flèche).

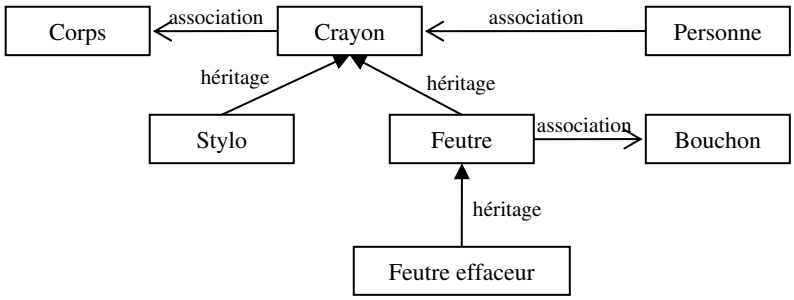



**Figure 12 : Représentation simple sous forme de graphe du diagramme de l'exercice « Stylo et Feutre »**

L'avantage de cette représentation est qu'elle est calquée sur la représentation graphique manipulée à l'interface par le concepteur du diagramme. Les graphes résultants ont peu de sommets et d'arcs. Elle a néanmoins quelques inconvénients :

- **des éléments UML ne peuvent être transcrits dans cette représentation** : une relation n-aire ne peut pas être représentée dans ce formalisme étant donné qu'un arc d'un graphe est binaire. Cette même remarque s'applique à la classe d'association car elle serait à la fois un sommet et un arc du graphe ;
- **certaines étiquettes ne sont pas atomiques** : une caractéristique UML (i.e. les opérations, les attributs et les fins d'association) n'est pas une étiquette atomique comme l'est un type, une visibilité ou un nom. Chaque caractéristique UML a notamment des étiquettes la décrivant. Un attribut peut par exemple avoir un nom, un type, une valeur par défaut, une visibilité.

Cette représentation est incomplète car elle induit des relations implicites de propriété entre les classificateurs et leurs caractéristiques (et également entre les relations d'association et leurs extrémités).

### 1.4.2.2 Représentation arborescente XMI

Une représentation sous forme d'arborescence de conteneurs (et de contenus) précise les liens de propriété entretenus entre les classificateurs, les relations et les caractéristiques UML.

```
- <Model>
| - name : "Solution2_Stylo_Feutre"
|
| - <Class>                          // Classe abstraite Crayon
| | - name : "Crayon"
| | - isAbstract : true
| | - <Property>                     // Attribut couleur de type chaine de caractère
| | | - name : "couleur"
| | | - type : String
| | | - visibility : protected
| | - <Property>                     // Attribut marque de type chaine de caractère
| | | - name : "marque"
| | | - type : String
| | | - visibility : protected
| | - <Property>                     // Fin d'association de "possède" du coté de la classe "Corps"
| | | - type : Corps
| | | - visibility : private
| | | - aggregation : composite
| | | - association : possède
| | | - lowerValue : 1
| | | - upperValue : 1
| | - <Property>                     // Fin d'association de "appartient à" du coté de la classe "Personne"
| | | - name : "propriétaire"
| | | - type : Personne
| | | - visibility : private
| | | - aggregation : none
| | | - association : appartient à
| | | - lowerValue : 1
| | | - upperValue : 1
| | - <Property>                     // Fin d'association de "est utilisé par" du coté de la classe "Personne"
| | | - name : "utilisateur"
| | | - type : Personne
| | | - visibility : private
| | | - aggregation : none
| | | - association : est utilisé par
| | | - lowerValue : 0
| | | - upperValue : 1
| | - <Operation>                    // Opération écrire de la classe Crayon
| | | - name : "écrire"
|
| - <Class>                          // Classe Feutre
| | - name : "Feutre"
| | - <Property>                     // Fin d'association de "possède" du coté de la classe "Bouchon"
| | | - type : Bouchon
| | | - visibility : private
| | | - aggregation : shared
| | | - association : possède
| | | - lowerValue : 0
| | | - upperValue : 1
| | - <Operation>                    // Opération boucher de la classe Feutre
| | | - name : "boucher"
| | - <Operation>                    // Opération déboucher de la classe Feutre
| | | - name : "déboucher"
| | - <Generalization>               // Relation d'héritage vers la classe mère Crayon
| | | - general : Crayon
...
| - <Class>                          // Classe Personne
| | - name : "Personne"
| | - <Property>                     // Attribut nom de type chaine de caractère
| | | - name : "nom"
| | | - type : String
| | | - visibility : private
| | - <Property>                     // Fin d'association de "appartient à" du coté de "Crayon"
| | | - type : Crayon
| | | - visibility : private
| | | - aggregation : none
| | | - association : appartient à
| | | - lowerValue : 0
| | | - upperValue : *
| | - <Property>                     // Fin d'association de "est utilisé par" du coté de la classe "Crayon"
| | | - type : Crayon
| | | - visibility : private
| | | - aggregation : none
| | | - association : est utilisé par
| | | - lowerValue : 0
| | | - upperValue : 1
...
| - <Association>                    // Association "possède" entre "Feutre" et "Bouchon"
| | - name : "possède"
| | - memberEnd : <Property> - : Bouchon [0..1], <Property> feutre : Feutre [0..1]
|
| - <Association>                    // Association "est utilisé par" entre "Personne" et "Crayon"
| | - name : "est utilisé par"
| | - memberEnd : <Property> utilisateur : Personne [0..1], <Property> - : Crayon [0..1]
|
| - <Association>                    // Association "appartient à" entre "Personne" et "Crayon"
| | - name : "appartient à"
| | - memberEnd : <Property> propriétaire : Personne [1..1], <Property> - : Crayon [0..*]
```

**Figure 13 : Extrait du diagramme de l'exercice « Stylo et Feutre » sous forme d'arborescence XMI**

Une représentation arborescente est adoptée par exemple dans le format XMI (encodant un modèle UML en XML). Un diagramme est représenté sous la forme d'un arbre où la racine est un élément de type *Model* (le diagramme). Les nœuds de l'arbre sont les éléments UML hiérarchisés par des liens de propriété (i.e. contenance). Dans ce format, un modèle (sans paquetage) est propriétaire des classes et des relations d'association qui le compose. Les classes sont ensuite propriétaires des attributs et des opérations qu'elles encapsulent et des relations d'héritage les raccordant à leurs classes mères. Les terminaisons d'association peuvent être possédées soit par les relations d'association soit par les classes liées. La figure 13 représente un extrait du diagramme de classes modélisé pour l'exercice « Stylo et Feutre ». Pour une meilleure lisibilité, nous avons filtré les balises XML (cf. annexe 4) et nous n'avons pas reporté certaines valeurs pour la visibilité et l'abstraction lorsqu'elles sont respectivement *public* et *false* car ce sont leurs valeurs par défaut.

La représentation sous forme d'arbre est plus complète que celle définie à partir de la représentation graphique. Néanmoins, certains liens structurels entretenus entre les relations d'association, les fins d'association et les classes ne sont pas accessibles directement : c'est en consultant les propriétés possédées par les classes ou les relations d'association que l'on peut connaître les liens entretenus entre une association et plusieurs classes du diagramme (cf. les champs « association » des éléments de type « Property » dans la figure 13). Cette spécificité découle directement du métamodèle UML 2.x car les extrémités d'association et les attributs sont des éléments conceptuellement très proches[8]. C'est pour cela que ces deux éléments sont regroupés sous le terme de *propriété* (une caractéristique structurelle dans la représentation interne des diagrammes UML). Dans le cas d'une classe, les propriétés sont les attributs et les éventuelles terminaisons d'association que possède la classe. Dans le cas d'une association, les propriétés sont constituées par les terminaisons d'association que possède l'association (une association n-aire peut en avoir plus de deux). L'extrait suivant du métamodèle UML pour le diagramme de classes met en avant le rôle structurant des propriétés dans les diagrammes de classes UML.

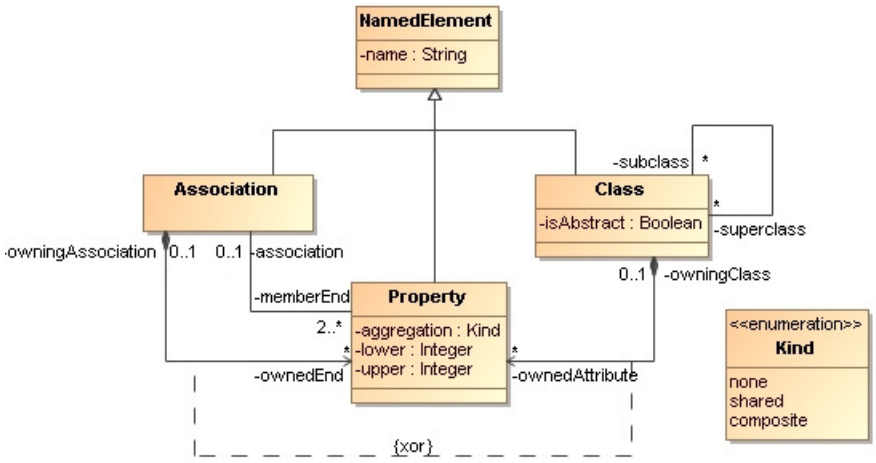



**Figure 14 : Extrait du métamodèle UML 2.x centré sur les propriétés**

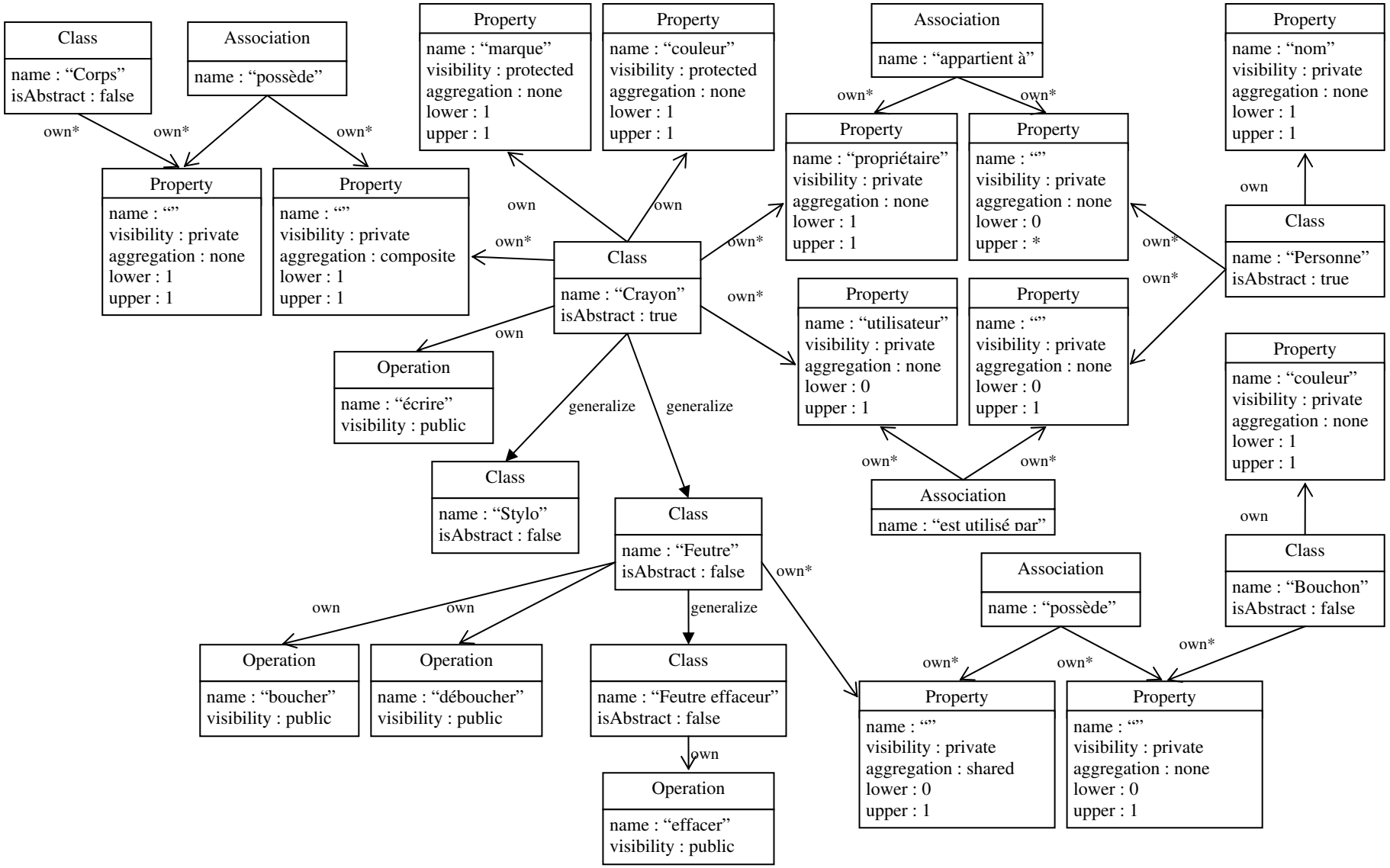
[8] Un attribut peut être considéré comme une association dégénérée dans laquelle une terminaison d'association est détenue par un classificateur (généralement une classe). Le classificateur détenant cette terminaison d'association devrait théoriquement se trouver à l'autre extrémité, non modélisée, de l'association. Un attribut n'est donc rien d'autre qu'une terminaison d'un cas particulier d'association [Audibert 2008].

#### 1.4.2.3 Représentation sous forme de graphe UML

En s'appuyant sur cet extrait du métamodèle UML, nous pouvons transformer un diagramme de classes UML en graphe UML où tous les éléments UML et leurs liens directs sont rendus explicites à l'aide de nœuds et d'arcs. Comme dans la représentation sous forme d'arbre, la majorité des éléments de modélisation (classificateurs, associations, attributs, fins d'association et opérations) sont des sommets du graphe. Les arcs sont les héritages, les relations de propriété et de typage entre les différents éléments. Une classe (un sommet) possède (un arc) des attributs (des sommets) qui peuvent être typées (arc) par des classes. Les associations possèdent des extrémités d'association (sommets) qui peuvent être typés (arc) par des classes (sommets). Le schéma reporté dans la figure 15 est un graphe du même diagramme UML que ceux des précédentes représentations. Nous n'avons pas reproduit les relations de typage entre les éléments pour ne pas surcharger la lecture du graphe. De plus, pour simplifier le problème de possession des fins d'association (ce point est ambigu et est géré de différentes manières dans les éditeurs UML), nous avons reporté des liens de propriété (noté *own**) entre les classes et les associations liés à une fin d'association.

Les dernières représentations sous forme d'arbre et de graphe expriment clairement les liens entretenus au niveau de leurs éléments et de leurs caractéristiques dans les diagrammes UML. Chaque arc relie deux nœuds et est doté d'une direction. Les nœuds et les arcs sont caractérisés au moyen d'un type et d'étiquettes représentant leurs attributs (nom, visibilité, abstraction, agrégation, multiplicités). Ces représentations correspondent naturellement aux spécifications décrites dans le métamodèle d'UML. Les représentations de graphes des trois cas exposés sont équivalentes sémantiquement et il est possible de passer des unes aux autres en leur appliquant des transformations.

Nous reviendrons sur l'utilisation de ces représentations lors de la présentation de notre méthode d'appariement de diagrammes (cf. chapitre 5).




**<u>Figure 15 :</u> Représentation du diagramme de l’exercice « Stylo et Feutre » sous forme d’un graphe UML**

# CHAPITRE 2

# Les environnements dédiés à l'apprentissage de la modélisation

# PLAN DU CHAPITRE

Nous avons montré précédemment que la modélisation tient une place importante en informatique lors du développement d'un logiciel. Des outils graphiques performants sont utilisés depuis quelques années pour construire et faire évoluer les modèles tout au long d'un projet de génie logiciel mené en entreprise. Il existe notamment de nombreux outils commerciaux, gratuits ou open-source utilisant UML et répondant aux besoins des entreprises. Ces outils sont très souvent centrés sur le code et peuvent aller du simple éditeur de modèles à de véritables ateliers de génie logiciel. Nous pouvons citer les outils professionnels suivants : IBM Rational Rose [Rational], Objecteering [Objecteering], Borland Together [Together], MagicDraw UML [MagicDraw], Poséidon [Poseidon], ArgoUML [ArgoUML]. Objecteering, Borland Together et IBM Rational Rose sont les plus répandus et les plus sophistiqués car ils constituent de véritables ateliers de génie logiciel très complets en termes de fonctionnalités et d'outils mis à la disposition à la fois des analystes, des concepteurs et des programmeurs. L'objectif principal de ces environnements est de permettre à une équipe de concevoir des applications orientées objet de qualité et d'en accélérer le processus de développement. Pour cela, ils proposent une vaste gamme de produits, d'outils et de services embarqués tels que le support de la majorité des éléments et des diagrammes UML en respectant au maximum le métamodèle d'UML défini par l'OMG, l'exportation et/ou l'importation de modèles dans différents formats standards, la génération de code dans un ou plusieurs langages de programmation cibles, l'ingénierie inverse des constructions permettant de répercuter automatiquement des modifications du code sur les modèles construits et également la gestion des versions des modèles.

Les environnements de modélisation et de développement cités sont conçus et adaptés pour un usage professionnel pratiqué par des adeptes et des experts. Bien qu'ils n'aient pas été conçus initialement dans un but d'apprentissage de la modélisation, leur utilisation dans la pratique pour des besoins d'enseignement est répandue, ce qui soulève certains problèmes. Même dans leurs versions mises à disposition pour un usage personnel ou d'enseignement, qui sont plus simples et limitées, ils demeurent trop complexes pour des besoins d'enseignement initial. Ils peuvent notamment amener des confusions chez les apprenants débutants car ces environnements utilisent la syntaxe complète d'UML pour représenter les éléments [Ramollari & Dranidis 2007] et offrent un très grand nombre de fonctionnalités surchargeant l'activité d'enseignement. Leur utilisation demande donc un apprentissage propre qui peut constituer un obstacle lorsqu'il est couplé à l'apprentissage des concepts de la modélisation. Selon certains auteurs, un sous-ensemble simplifié de la notation d'UML est suffisant pour enseigner la conception de systèmes orientés objet [Ramollary & Dranadis 2007].

La grande majorité des environnements dédiés purement à l'enseignement de la modélisation adaptent la complexité aux besoins d'enseignement par une **simplification** des éléments pouvant poser un problème. Pour que l'environnement soit intuitif et simple à utiliser, la notation, l'interaction, les fonctionnalités et les outils peuvent être réduits à leur strict nécessaire afin que l'environnement de travail ne constitue plus un obstacle pour l'apprenant. À partir de ce point commun, les environnements d'enseignement de la modélisation peuvent se concentrer sur des approches, des particularités ou des fonctions différentes permettant de faciliter l'enseignement et l'acquisition de bonnes pratiques dans le cadre de l'activité de modélisation. Certains environnements se concentrent sur la réalisation de diagrammes corrects et cohérents entre eux en mettant l'accent notamment sur des dispositifs automatiques de vérification de la cohérence des modèles construits. D'autres peuvent orienter l'interaction, porter un jugement sur les actions de l'apprenant et lui proposer des explications, c'est le cas des systèmes tutoriels intelligents. Enfin les derniers mettent l'accent sur le fait que l'activité de modélisation se déroule en équipe. Ils supportent et encouragent donc la

collaboration entre les apprenants pendant l'activité de modélisation. Nous présentons dans la suite de ce chapitre des exemples d'environnements d'enseignement de la modélisation mettant l'accent sur chacun de ces points en définissant à chaque fois leur contexte, leur objectif, leur public cible, leur démarche et leur fonctionnement général. La manière dont ces environnements analysent les productions des apprenants sera présentée en détail ultérieurement au chapitre 3. Nous décrivons enfin l'environnement Diagram dans lequel nos travaux s'inscrivent.

## 2.1 Systèmes centrés sur la vérification de la cohérence des modèles

Étant donné que de nombreux développeurs dans un projet informatique peuvent produire et mettre à jour des documents et des modèles, les informations contenues peuvent être ou devenir incohérentes. Les incohérences sont indésirables et à bannir autant que possible car elles causent de mauvaises interprétations pouvant entraîner des conséquences plus ou moins sérieuses sur le cycle de développement d'une application [Lange & Chaudron 2006]. En situation d'apprentissage, les modèles produits par les étudiants souffrent aussi de ce type de problème qui survient du fait d'un manque de coordination du groupe, d'un manque d'attention, mais également d'un manque de connaissances [Ramollari & Dranidis 2007]. Les éditeurs professionnels se focalisent peu sur ces problèmes, probablement parce qu'ils ne les considèrent pas à proprement parler comme des erreurs mais comme des redondances ou des omissions. La vérification de la cohérence des modèles est donc un problème qu'il faut traiter au plus tôt au cours de l'apprentissage de la modélisation, notamment avec UML.

### *2.1.1 La vérification de la cohérence des diagrammes UML*

Un « projet UML » consiste en un ensemble de diagrammes qui, mis ensemble, décrivent un seul système. Les modèles UML sont **incohérents** lorsqu'un ou plusieurs éléments d'UML y sont employés en transgressant les relations sémantiques qui lient ces éléments [Malgouyres 2006]. Ces relations sémantiques peuvent être exprimées sous forme de règles de cohérence ou règles de bonne formation (*well-formedness rules*) qui expriment l'usage correct des constructions et non pas la fonction que les constructions peuvent remplir dans une modélisation. Les incohérences reposent sur la **sémantique de vérification** des constructions du langage UML (la bonne façon de constituer un modèle UML) et non sur leur **sémantique opérationnelle** (leurs apports descriptifs dans un modèle). Une **incohérence** est donc la violation d'une propriété associée au langage UML et qui doit être respectée pour tout modèle UML [Seuma Vidal 2006].

La vérification de la cohérence peut être réalisée suivant deux angles : d'une part, la conformité interne d'un diagramme (*internal diagram correctness*) et d'autre part la cohérence inter-diagrammes (*inter-diagram consistency*).

La **validation de la conformité** permet de repérer tous les problèmes de non-conformité au métamodèle, aux spécifications et à la syntaxe du langage de modélisation adopté dans un modèle indépendamment des autres. Elle peut être considérée comme une sorte de « validation syntaxique » au niveau du modèle construit. Elle permet de mettre en avant des problèmes de « forme » du modèle qui peuvent avoir ou non des conséquences sur le sens des éléments représentés (le fond). Cette validation peut se faire directement à l'interface ou/et à la demande de l'utilisateur. Au regard de la **conformité interne d'un diagramme**, les outils peuvent ainsi limiter l'utilisateur dans la spécification de

diagrammes incorrects sans compromettre la flexibilité. Les éditeurs devraient notamment ne pas autoriser l'usage d'un type incorrect de relation dans la conception d'un diagramme de classes UML, comme lier par une relation de dépendance de réalisation une classe à une autre classe au lieu de la lier à une interface, ou définir une interface comme une agrégation de classes. Un exemple de 650 règles de cohérence internes aux modèles UML 2.0 a été exprimé dans [Seuma Vidal *et al.* 2005] en langage naturel (français) pour ne pas être limité par un quelconque langage formel.

La cohérence inter-diagramme est à la fois le problème le plus important et le plus négligé par les outils dédiés à la modélisation. La cohérence peut être soit **horizontale** en se focalisant sur des modèles de différentes parties d'un système soit **verticale** en concernant des modèles représentant la même partie d'un système à différents niveaux d'abstraction. Un dernier type de cohérence peut exister également entre différentes versions d'une collection d'artefacts d'un même système. La cohérence verticale est la moins spécifique à vérifier du fait qu'elle ne dépend ni du processus, ni de la méthode de développement adoptée. Nous ne nous intéressons pas dans nos travaux à la cohérence inter-diagramme du fait que les apprenants construisent un seul diagramme dans les activités d'apprentissage de la modélisation mises en place.

### 2.1.2 StudentUML

Actuellement, l'environnement StudentUML dédié à l'apprentissage de la modélisation avec UML apporte une réponse pertinente au problème de vérification de la cohérence des diagrammes UML produits par des apprenants en étant indépendant des processus et des méthodes de développement. StudentUML est présenté dans [Ramollari & Dranidis 2007] et a été évalué dans [Dranidis 2007].

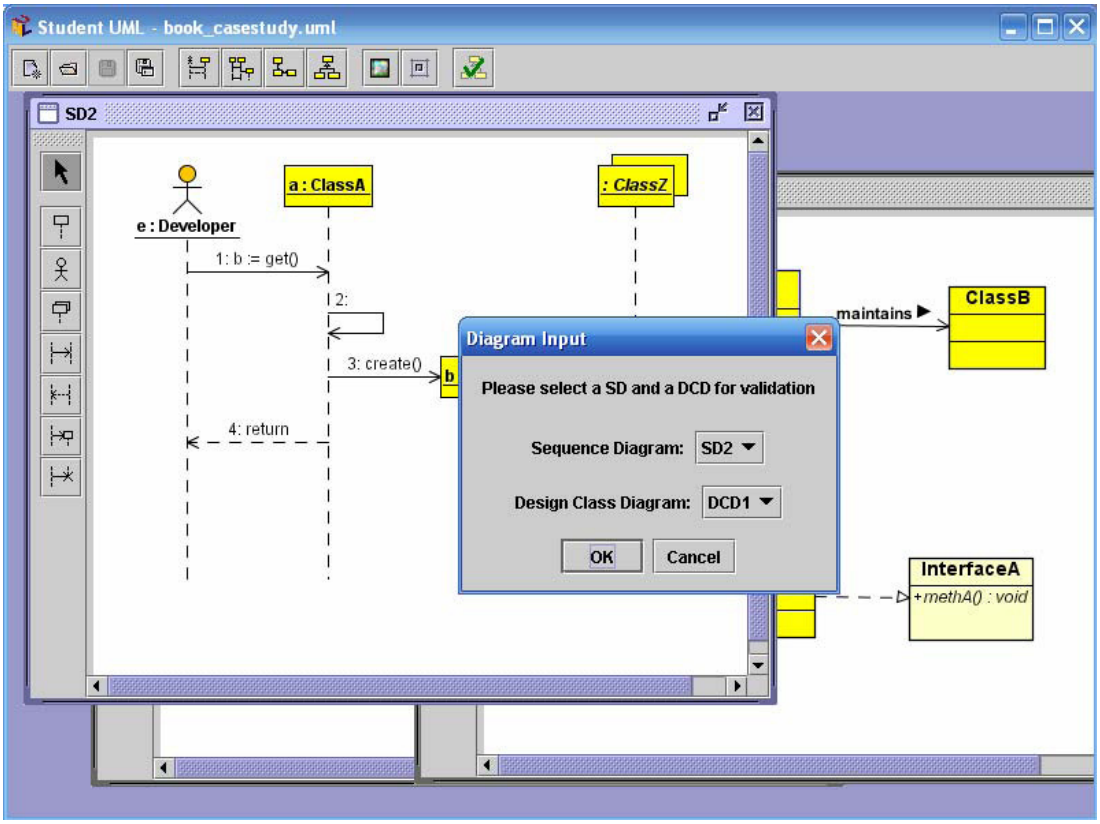



**Figure 16 : Interface graphique de StudentUML [Ramollari & Dranidis 2007]**

StudentUML repose sur deux principes forts : d'une part il est **simple** d'utilisation et de prise en main car il fournit un outil intuitif et une interface graphique épurée avec peu de menus, une syntaxe UML simplifiée et il limite l'utilisation des types d'éléments UML à ceux nécessaires dans une activité donnée (cf. figure 16). D'autre part, StudentUML apporte une vérification automatique de la cohérence intra et inter diagrammes complète à un même niveau d'abstraction.

Le résultat de la vérification de la cohérence est présenté sous forme de messages d'erreurs et d'avertissements où l'utilisateur a à sa disposition des options de correction proposées automatiquement par le système. Le système apporte également des conseils sur la manière dont les différents diagrammes d'un même projet doivent être en relation et comment éviter l'apparition d'incohérences. Il supporte enfin la conversion automatique d'un diagramme d'un type vers un autre type dans un même projet et permet de générer du code en langage objet (Java ou C++) à partir des diagrammes produits. La synchronisation des diagrammes en fonction des modifications du code et inversement se font automatiquement.

La figure 17 montre un exemple de résultats de vérification de la cohérence du diagramme de classes « DCD1 » et du diagramme de séquence « SD2 » présents dans le projet de la figure 16. Deux avertissements concernent la validité interne du diagramme de séquence (l'objet issu de la classe « ClassZ » et le message « Call Message 2 » ne sont pas nommés). L'erreur et le dernier avertissement se focalisent sur la vérification de la cohérence inter-diagramme. L'erreur met en avant l'omission d'un élément dans le diagramme de classes : la classe « ClassZ » n'est pas représentée dans le diagramme de classes « DCD1 » alors qu'elle existe dans le diagramme de séquence. L'option de correction est fournie dans ce message à l'apprenant qui pourra ajouter l'élément manquant au diagramme de classes pour maintenir la cohérence du projet.

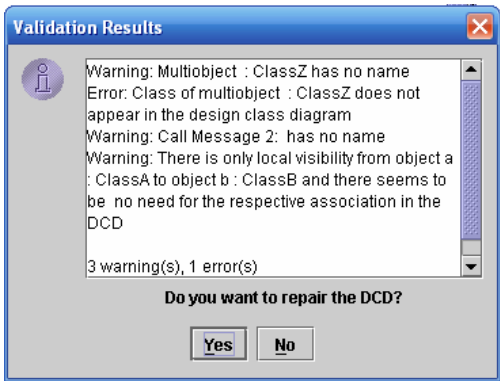



**Figure 17 : Exemples d'avertissements et d'erreurs dans StudentUML [Ramollari & Dranidis 2007]**

Cet environnement d'apprentissage présente l'avantage d'être ouvert car il permet de construire librement les diagrammes d'un projet sans contraintes prédéfinies. StudentUML a la capacité de vérifier la cohérence des diagrammes construits mais les auteurs ne précisent pas la méthode de validation utilisée pour effectuer ce contrôle et fournir ensuite des rétroactions et des options de correction automatique. Aucune solution ou base d'erreurs n'est prédéfinie pour les exercices à réaliser et ainsi aucun contrôle du sens modélisé dans les diagrammes n'est possible dans cet environnement.

## 2.2 Systèmes tutoriels intelligents

La vérification de la cohérence s'attaque à des problèmes de « forme » des modèles alors que les STI (**S**ystèmes **T**utoriels **I**ntelligents) mettent l'accent sur ce que l'apprenant veut représenter (le « fond » ou sens véhiculé par le modèle) et la manière dont il le fait (ses actions). Les tutoriels intelligents peuvent être vus comme des systèmes experts pour l'enseignement [Py 1996]. Ce sont des systèmes actifs, qui peuvent orienter l'interaction, porter un jugement sur les actions de l'apprenant et lui proposer des explications. Pour cela, ils possèdent en interne des connaissances sur le domaine permettant de savoir résoudre le problème posé à l'élève, des connaissances sur l'élève pour savoir analyser la réponse fournie et enfin des connaissances sur la pédagogie pour pouvoir expliquer et guider l'élève. KERMIT et DesignFirst-ITS sont deux exemples de tutoriels intelligents apportant des aides individualisées sur les erreurs que commettent les apprenants au cours de l'activité de modélisation. Pour traiter cet objectif commun, ces systèmes reposent sur deux approches différentes : des bases de contraintes pour KERMIT, un modèle du *curriculum* et un générateur semi-automatique de diagramme idéal à partir d'un énoncé exprimé en langage naturel pour DesignFirst-ITS.

### *2.2.1 KERMIT*

KERMIT (*a* ***K****nowledge-based* ***E****ntity* ***R****elationship* ***M****odelling* ***I****ntelligent* ***T****utor*) de [Suraweera & Mitrovic 2002] [Suraweera & Mitrovic 2004] est un environnement de résolution de problèmes conçu pour des étudiants d'université apprenant à modéliser des bases de données conceptuelles. Il permet notamment d'enseigner les bases de la modélisation Entité-Relation et il est considéré comme un complément de cours où il est supposé que l'apprenant connaît les fondamentaux en bases de données. Le système propose un problème à l'apprenant qui doit, via une interface adaptée, modéliser un schéma Entité-Relation satisfaisant l'énoncé. KERMIT a été développé en Microsoft Visual Basic et repose sur l'éditeur Microsoft Visio pour construire les modèles Entité-Relation.

L'interface graphique de KERMIT, illustrée dans la figure 18, est divisée en trois parties. Tout d'abord, l'énoncé du problème à modéliser est représenté dans la partie haute de l'interface. Ensuite, la partie centrale est constituée d'une part d'une barre d'outils contenant les éléments graphiques pouvant être utilisés dans un modèle Entité-Relation et d'autre part le modèle Entité-Relation en cours de construction par l'apprenant. Enfin, dans le bas de l'interface, un assistant virtuel et une zone d'affichage permettent de fournir des conseils et des rétroactions pédagogiques à l'apprenant au cours de l'activité de modélisation.

KERMIT force les étudiants à se focaliser sur l'énoncé du problème pour qu'ils élaborent une solution complète du problème. Pour la modélisation de chaque élément, l'assistant virtuel de l'environnement demande notamment à l'élève de surligner l'expression correspondante dans le texte. Dans l'énoncé, l'expression prend un style différent en fonction du type d'éléments qu'il représente dans le modèle (gras et bleu pour les entités, gras et vert pour les relations, italique et rose pour les attributs...).

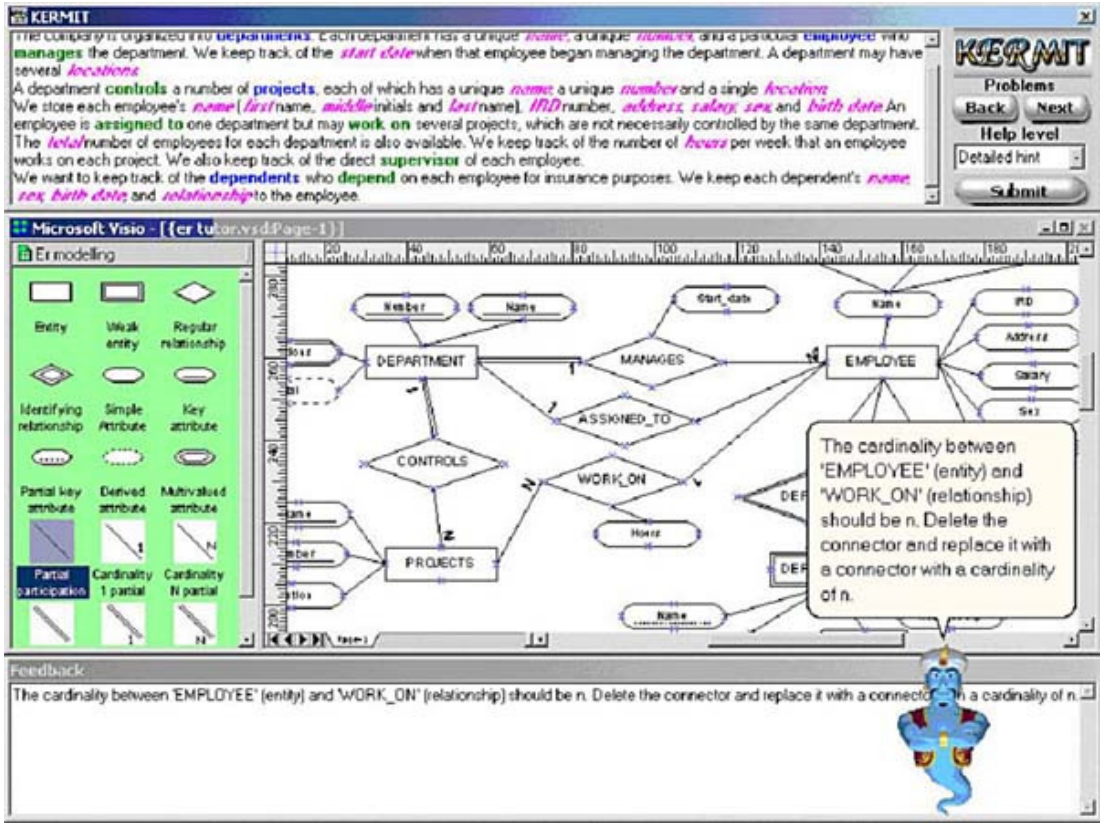



**Figure 18 : Interface graphique de KERMIT [Kermit]**

KERMIT repose sur l'approche CBM (***C**onstraint-**B**ased **M**odelling*) [Ohlsson 1994] et sur l'emploi d'une solution idéale pour modéliser le domaine de connaissances et générer un modèle de l'apprenant. Cette approche se concentre sur les violations des principes de base du domaine d'enseignement et permet de ne pas constituer de bibliothèque d'erreurs et de module expert. La base de connaissances de KERMIT est composée d'une centaine de contraintes syntaxiques et sémantiques. Les contraintes syntaxiques décrivent les schémas Entité-Relation valides syntaxiquement et sont utilisées pour identifier les erreurs syntaxiques dans la solution de l'apprenant. Les contraintes sémantiques, généralement plus complexes que les contraintes syntaxiques, décrivent les écarts autorisés entre la solution de l'apprenant et une solution idéale. La mise en correspondance des noms est un problème important lors de l'évaluation de la solution de l'apprenant[9]. Ce problème est traité dans KERMIT en obligeant l'étudiant à sélectionner dans l'énoncé le terme qui sera modélisé par une construction du modèle Entité-Relation.

Pour s'adapter à différents étudiants, KERMIT maintient par recouvrement un modèle à long terme pour chaque apprenant (la sélection de problèmes et les rétroactions). Un modèle de l'apprenant contient des informations sur l'historique des sessions précédentes (comme une liste des problèmes correctement résolus) et contient également un modèle de la connaissance de l'étudiant, exprimé en termes de contraintes.

[9] Étant donné qu'il n'y a pas de standards qui forcent le nommage des entités, des relations ou des attributs, les étudiants peuvent utiliser des synonymes, des mots, des phrases similaires pour nommer une construction particulière de leur modèle. Les noms des constructions dans la solution de l'apprenant peuvent alors ne pas s'apparier à ceux de la solution idéale et il est difficile dans ce cas de trouver une correspondance entre les constructions [Suraweera & Mitrovic 2002].

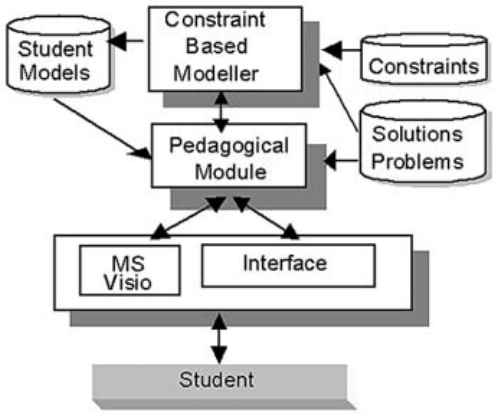



**Figure 19 : Architecture de KERMIT [Kermit]**

Le module pédagogique de KERMIT repose sur six niveaux de rétroactions pour assister l'apprenant au cours de l'activité de modélisation :

- **Correcte** : le premier niveau indique simplement à l'apprenant si sa solution est correcte ;
- **Drapeau sur une erreur** : le second niveau montre à l'apprenant une erreur qu'il a faite ;
- **Conseil et conseil détaillé** : le troisième et le quatrième niveau de rétroaction proposent un conseil plus ou moins détaillé à l'apprenant pour qu'il corrige une de ses erreurs ;
- **Toutes les erreurs** : le cinquième niveau met en avant toutes les erreurs commises par l'apprenant ;
- **La solution** : le dernier niveau donne la solution idéale à l'apprenant.

Les conseils et les rétroactions sont apportés à l'apprenant à sa demande sous forme de messages textuels par un assistant virtuel et sont reportés également dans la zone des rétroactions situés en bas de l'interface de KERMIT (cf. figure 18). L'apprenant peut choisir lui-même le niveau de rétroactions en fonction de ses besoins.

KERMIT est un environnement ouvert d'apprentissage de la modélisation de bases de données prodiguant un tutorat individuel sous forme de rétroactions pédagogiques aux apprenants pendant l'activité de modélisation. Chaque rétroaction est construite ici à partir de la violation d'une contrainte (une erreur commise par l'apprenant). Cet environnement présente l'avantage de permettre la manipulation du texte du problème tout au long de l'activité mais il contraint l'édition des éléments par rapport à l'énoncé. En effet, l'apprenant n'a pas de réelles possibilités pour représenter des éléments non spécifiés explicitement dans l'énoncé. L'enseignant, quant à lui, peut ajouter de nouveaux exercices en définissant l'énoncé et une solution idéale correspondant dans une interface dédiée à l'enseignant. L'utilisation de l'environnement est restreinte aux novices et les auteurs préconisent de pas introduire d'éléments implicites dans l'énoncé et d'adapter les énoncés pour qu'ils contiennent le moins d'ambiguïtés possibles.

### *2.2.2 DesignFirst-ITS*

Dans le contexte d'un CS1 (cours d'introduction à la programmation orientée objet de premier semestre), CIMEL ITS (***C****onstructive, collaborative* ***I****nquiry-based* ***M****ultimedia* ***E-L****earning*) [Moritz 2008] [Parvez 2007] [Wey 2007], renommé en DesignFirst-ITS, est un système tutoriel intelligent qui fournit un tutorat individuel dans le but d'aider des novices à apprendre les concepts de l'analyse et de la conception orientée objet en utilisant les éléments

d'UML. L'introduction des concepts d'objets, de classes et d'instances avant les éléments procéduraux d'un langage de programmation est réalisée dans l'approche *« Objects-First »* insistant plutôt sur le codage que sur la résolution de problème ou la conception. DesignFirst-ITS repose sur l'approche *« Design-First »* [DesignFirst], qui subsume *« Objects-First »* en leçons et introduit ainsi l'analyse et la conception orientée objet en se focalisant en plus sur des techniques de résolution de problèmes et en utilisant les éléments UML avant d'implanter du code. Les étudiants apprennent donc ici à comprendre et à résoudre un problème, puis à concevoir une solution en utilisant des méthodes et des outils modernes de conception logicielle sans se soucier de la syntaxe des langages de programmation.

DesignFirst-ITS supporte différents styles d'apprentissage et interagit avec les étudiants *via* deux systèmes. Le premier est un IDE (***I**ntegrated **D**evelopment **E**nvironment*), nommé LehighUML, développé sous forme d'un composant logiciel d'Eclipse [Eclipse] dans lequel les étudiants apprennent à modéliser un problème sous forme de diagrammes UML. LehighUML génère ensuite automatiquement le code en JAVA correspondant aux diagrammes créés. Ici, DesignFirst-ITS observe la progression de l'apprenant et l'assiste individuellement en se basant sur des stratégies pédagogiques. Le second système est un didacticiel multimédia nommé CIMEL qui introduit les concepts de la programmation orientée objet et de JAVA. Le STI peut renvoyer l'étudiant dans des portions de CIMEL pour qu'il révise les thèmes demandant un approfondissement. Il détermine également son niveau de compréhension des différents concepts à acquérir grâce à des questionnaires et des exercices interactifs fermés.

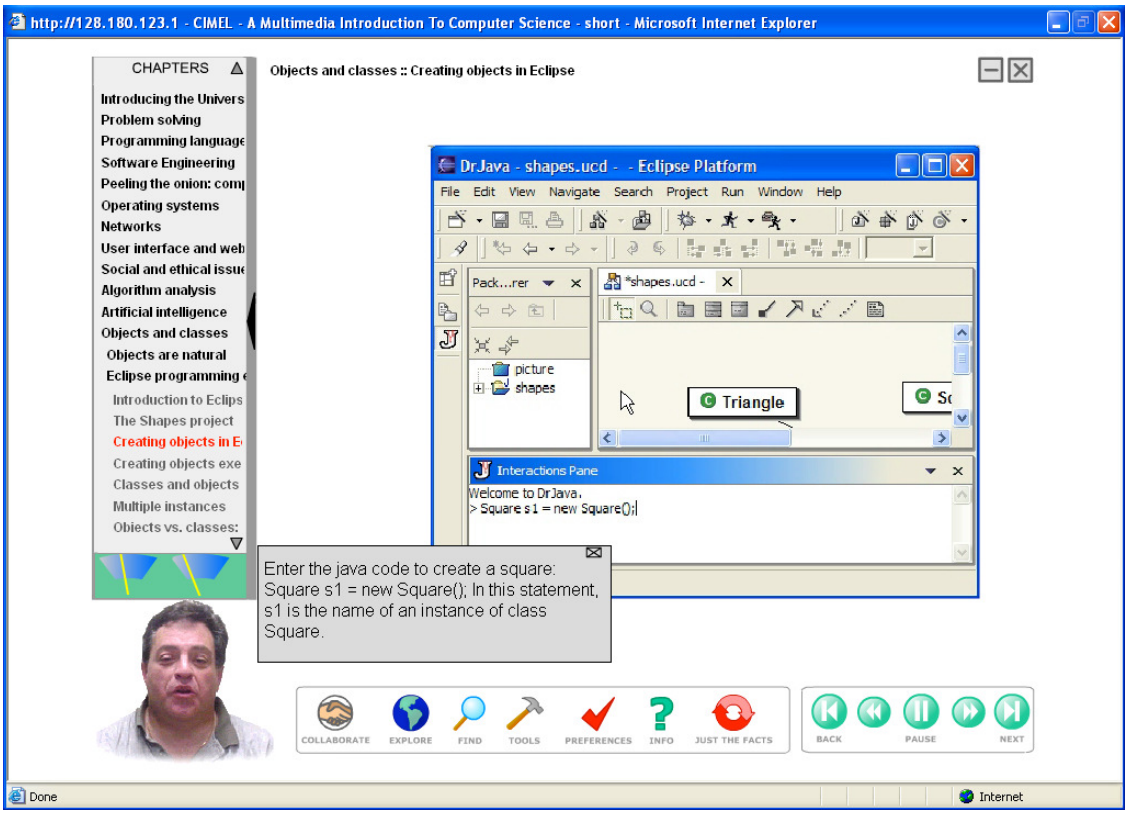



**Figure 20 : Interface graphique de CIMEL Multimedia [Moritz & Blank 2005]**

Les différents concepts à acquérir lors d'une activité sont abordés pas à pas par l'apprenant dans CIMEL. L'interface de l'environnement (cf. figure 20) permet de visualiser l'ensemble des chapitres et des activités à aborder. L'apprenant peut choisir de réaliser, de suivre une séquence d'activités à son rythme et de rejouer au besoin les activités

en fonction de ses difficultés. Il a accès à tout moment au cours et à un agent pédagogique qui lui fournit des explications et des aides tout au long de l'activité.

L'architecture de DesignFirst-ITS est divisée en cinq composants (cf. figure 21). Le cœur du système est le modèle du *Curriculum* (une représentation du modèle du domaine) organisé sous forme d'un CIN (***C**urriculum **I**nformation **N**etwork*) liant les concepts de la conception orientée objet ensemble pour montrer les relations entre eux (pré-requis et équivalences par exemple). Le second composant est un expert évaluateur qui joue le rôle d'interface entre l'éditeur LehighUML (troisième composant) et l'étudiant en observant ses actions tout au long de l'activité de modélisation. Ce composant compare pas à pas les actions de l'étudiant à une solution de référence prédéfinie par l'enseignant et génère un paquet de données pour une action correcte et un paquet d'erreur pour une action incorrecte. Ces paquets sont transmis au quatrième composant, le modèle de l'apprenant [Wey 2007], qui détermine le niveau de connaissance de l'apprenant pour chaque concept et essaie de découvrir les raisons des erreurs de l'apprenant. Enfin le conseiller pédagogique s'appuie sur les informations des autres composants pour déterminer le contenu et la forme des rétroactions à fournir à l'apprenant.

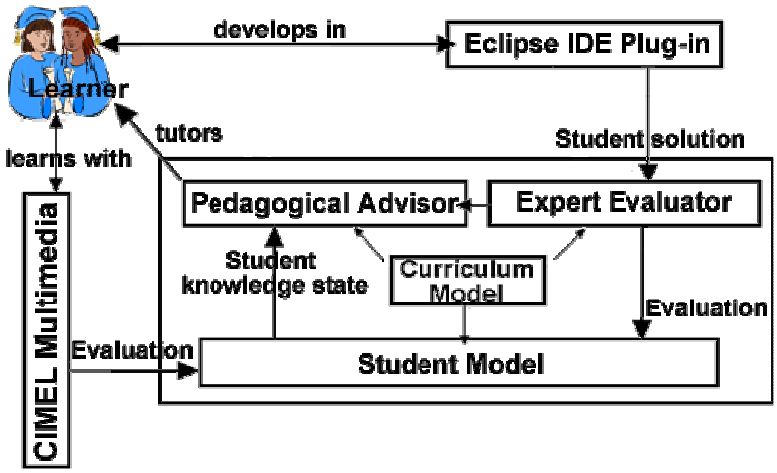



**Figure 21 : Architecture de DesignFirst-ITS [Parvez 2007]**

Le conseiller pédagogique [Parvez & Blank 2008] est basé sur le modèle psychologique de l'apprentissage de Felder-Silverman (*Felder-Silverman Learning Style Model*) [Felder & Silverman 1988]. Ce modèle classe les apprenants selon les moyens pertinents qu'ils utilisent pour percevoir l'information et traiter cette information [Boudreault *et al.* 2007]. Les types psychologiques selon le modèle de Felder-Silverman [Felder 1996] sont catégorisés en quatre dimensions opposant deux types à chaque fois : *sensoriel-intuitif*, *visuel-verbal*, *actif-réflexif*, et *séquentiel-global*. Ce modèle est transposé dans DesignFirst-ITS dans l'ILS (***I**ndex of **L**earning **S**tyle*) contenant les préférences de chaque apprenant. L'ILS est initialisé pour chaque individu à partir des réponses à un questionnaire (quarante-quatre questions) posé au début de la première activité. Le conseiller adapte donc la modalité d'interaction à chaque individu et lui prodigue des conseils et des explications plus ou moins détaillés. Le conseiller ne fournit jamais la solution à l'apprenant (car il n'existe pas une solution unique au problème).

L'environnement DesignFirst-ITS propose à la fois des exercices fermés et ouverts pour initier les novices aux concepts de la modélisation orientée objet. Tout comme KERMIT, les rétroactions pédagogiques se concentrent directement sur des erreurs que le système est capable d'identifier par rapport à une référence mais dans DesignFirst-ITS, les rétroactions sont adaptées individuellement à chaque apprenant en fonction du CIN, de l'historique

des actions (erronées rencontrées), et de leurs « préférences d'interaction ». La création et l'édition des diagrammes par les apprenants présentent l'avantage d'être peu contraintes dans l'environnement LehighUML, étant donné que l'environnement utilise différents composants externes capables d'analyser le langage naturel [Moritz 2008]. Ces systèmes peuvent identifier des synonymes des noms employés et corriger automatiquement certaines fautes de saisie et d'orthographe commises par les apprenants. L'enseignant peut enfin définir de nouveaux exercices de modélisation dans une interface guidant pas à pas la création de l'exercice. Un générateur de solution [Moritz 2008] produit automatiquement une solution pour un énoncé donné et assiste l'enseignant. L'inconvénient majeur est que les exercices produits sont très contraints car ils ne peuvent pas comporter de concepts implicites et leurs phrases doivent être formalisées de manière simple. Les exercices proposés laissent donc peu d'ouverture lors de l'activité de modélisation réalisée par l'apprenant.

## 2.3 Systèmes centrés sur l'apprentissage collaboratif

L'activité de modélisation en entreprise lors d'un projet de développement se déroule généralement en équipe ou en groupe. Les différents acteurs sont amenés à réfléchir et construire les modèles ensemble. L'apprentissage collaboratif présente un certain nombre d'avantages [Soller 2001] : il encourage les apprenants à « verbaliser » leur pensée, à travailler ensemble, à se poser des questions, à expliquer et à justifier leurs opinions. Il augmente à la fois la responsabilité des étudiants par rapport à leur apprentissage personnel et la possibilité de résoudre et d'examiner des problèmes sous différents angles. Enfin, la collaboration les encourage à réfléchir sur leurs connaissances. Plusieurs environnements d'apprentissage de la modélisation se focalisent pour ces raisons sur les aspects collaboratifs de l'activité de modélisation en supportant et en encourageant la réflexion en groupe et la collaboration de différentes manières.

### *2.3.1 Modellingspace*

Modellingspace est un environnement d'apprentissage collaboratif de modélisation à distance [Komis *et al.* 2003] s'adressant à des élèves de 11 à 17 ans. Son but est la construction individuelle ou en équipe de modèles scientifiques dans le cadre de l'enseignement de diverses disciplines comme les mathématiques, la physique, la chimie, la biologie et dans le cadre d'approches pluridisciplinaires. L'apprentissage via cet environnement se fait de manière synchrone ou asynchrone par l'échange d'idées entre les différents membres de l'équipe. Le système permet notamment de créer des entités primitives, de créer, d'explorer et de tester des modèles. Cet environnement ne propose en revanche ni tutorat artificiel ni analyse des modèles conçus par les apprenants.

Modellingspace repose sur les mêmes concepts que le logiciel éducatif ModelsCreator [Politis *et al.* 2001], un environnement ouvert d'apprentissage individuel permettant aux élèves de concevoir et de modifier des modèles. Ces deux environnements de modélisation mettent un accent particulier sur le raisonnement qualitatif et semi-quantitatif ainsi que sur les moyens alternatifs d'expression et de visualisation des modèles. Ils permettent notamment à un élève de travailler à plusieurs niveaux conceptuels [Dimitracopoulou *et al.* 1999].

L'interface graphique de ModelsCreator (cf. figure 22) reprise en partie dans Modellingspace est constituée de deux grandes parties distinctes. La première est l'espace des « situations–problèmes » contenant les problèmes à modéliser et la seconde est l'espace de « la construction et de l'exploration du modèle » qui contient la zone de visualisation du modèle en construction, les outils de construction et l'espace d'exécution (simulation du comportement du modèle). Dans cette dernière, l'élève peut choisir les objets du modèle, régler leurs propriétés et les relier avec des relations appropriées (selon sa représentation mentale du modèle). Ensuite, il peut tester, modifier et suivre l'exécution du modèle à l'aide de plusieurs représentations multiples (simulations, diagrammes à barres, graphiques, tableaux de valeurs, tables de décision, etc.).

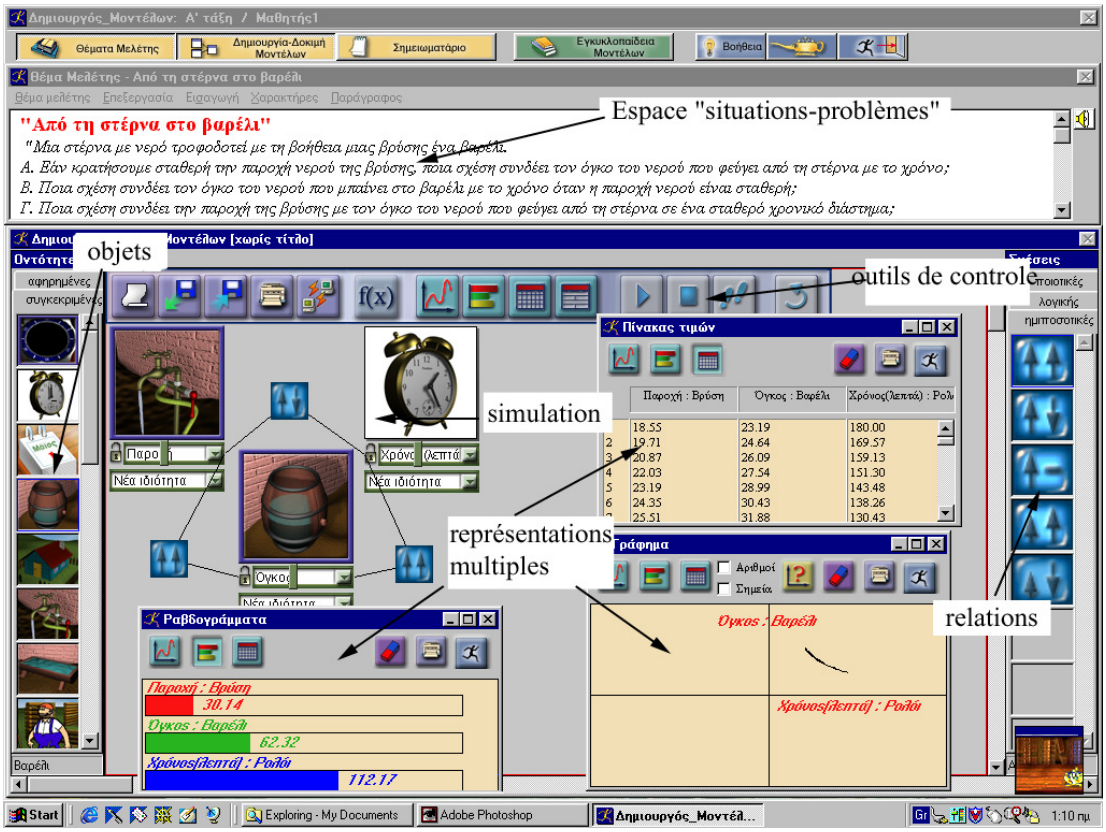



**Figure 22 : Interface de création et de simulation d'un modèle dans ModelsCreator [Politis *et al.* 2001]**

Modellingspace est muni également d'outils permettant aux enseignants de suivre le travail des élèves (analyse de discussion, système de supervision des collaborations...). Modellingspace est donc un outil favorisant cet échange et gérant les difficultés que peuvent rencontrer les apprenants lors de leur collaboration à distance. Cet environnement est un micromonde où l'apprenant peut très facilement élaborer des modèles, construire lui-même ses connaissances et interagit avec ses pairs. Néanmoins, l'apprenant n'a pas la possibilité ici de savoir si sa proposition est valide, cohérente ou erronée étant donné que le modèle n'est pas analysé comme c'est le cas dans les tuteurs intelligents.

### 2.3.2 Collect-UML

Dans le contexte de l'apprentissage des concepts de la modélisation orientée objet, l'équipe de Mitrovic propose un tutoriel intelligent individuel et collectif nommé Collect-UML [Baghaei & Mitrovic 2005] [Baghaei 2007]. C'est un environnement web de résolution de problèmes dans lequel des étudiants novices peuvent construire des diagrammes de classes UML qui satisfont un ensemble de spécifications décrites dans un énoncé textuel. Ce système est

conçu comme un complément à l'enseignement dispensé en classe et les étudiants sont donc déjà familiarisés avec les fondamentaux de la conception de systèmes orientés objet et d'UML.

La version de Collect-UML supportant l'apprentissage individuel des concepts de la MOO repose sur les mêmes principes que ceux définis dans KERMIT (cf. chapitre 2.2.1). L'énoncé est présent et manipulable à l'interface par l'apprenant (cf. figure 23). L'apprenant est toujours contraint d'utiliser les expressions de l'énoncé pour chaque élément de son diagramme. Des bases de contraintes relatives aux connaissances du domaine et une solution de référence sont utilisées pour l'évaluation de la validité de la solution proposée, le maintien du modèle de l'apprenant et la production de rétroactions pédagogiques individuelles.

Collect-UML se différencie de KERMIT dans sa version multi-utilisateurs. Collect-UML est ici un système CSCL (***C**omputer-**S**upported **C**ollaborative **L**earning*) qui a pour but d'apporter un soutien d'une part pour la résolution des problèmes et d'autre part pour l'interaction sociale entre les différents membres d'un groupe d'apprenants. La stratégie d'apprentissage collaborative utilisée dans Collect-UML est basée sur la théorie du conflit socio-cognitif (*socio-cognitive conflict theory*) [Doise & Mugny 1984]. Selon cette théorie, l'interaction sociale est constructive seulement si une confrontation est créée entre les solutions divergentes des étudiants. Le système a donc pour objectif de créer les conditions nécessaires pour que le conflit soit effectif. Pour cela, il identifie les différences entre la solution du groupe et les solutions individuelles, il rend les étudiants conscients de ces différences et il leur demande de résoudre les conflits en sollicitant et proposant des explications.

L'interface utilisée par les étudiants dans Collect-UML est illustrée dans la figure suivante :

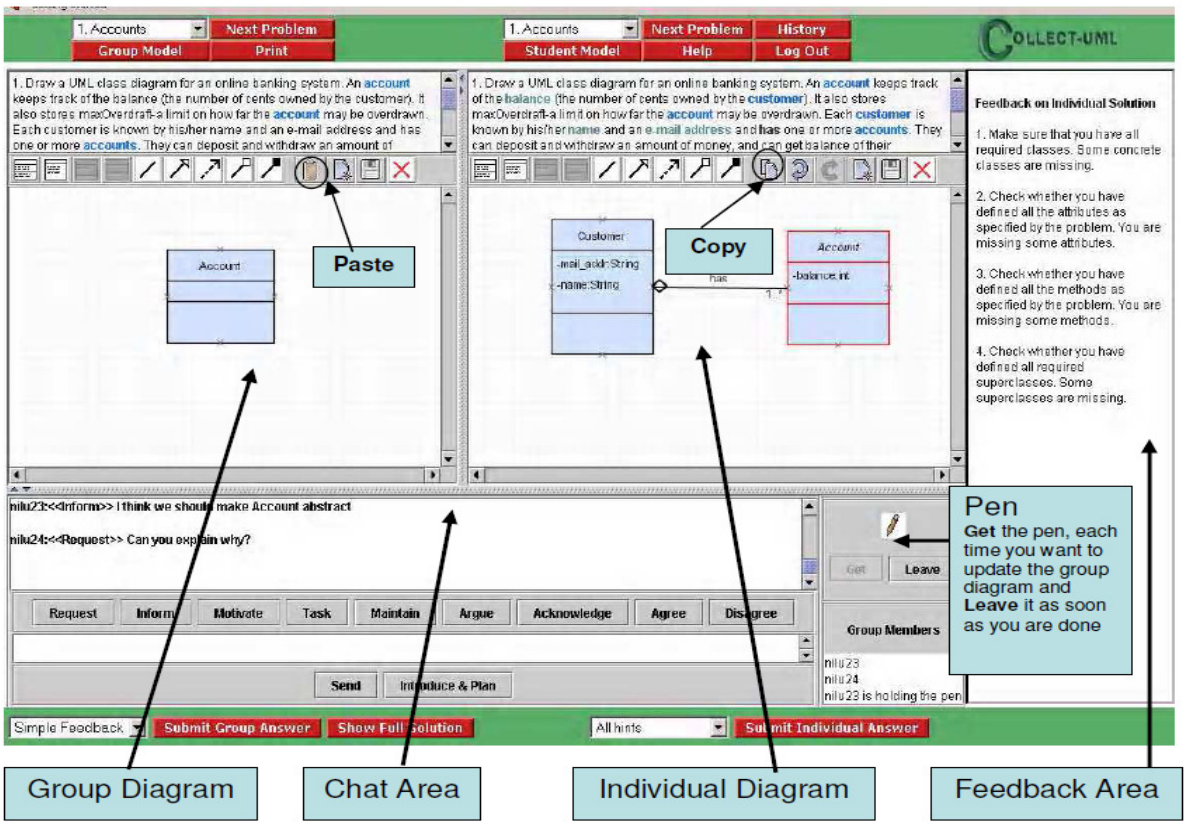



**Figure 23 : Interface graphique de la version multi-utilisateurs de Collect-UML [Baghaei 2007]**

La fenêtre de Collect-UML est séparée en deux : les étudiants construisent leurs solutions individuelles dans leur espace privé de travail (à droite de l'interface) et ils utilisent l'espace de travail partagé (à gauche de l'interface) pour construire les diagrammes collectifs en communiquant via la fenêtre de chat (en bas de l'interface). La fenêtre de groupe est initialement désactivée et devient active après un moment donné. Un seul apprenant peut alors mettre à jour le diagramme partagé à un instant donné en copiant/collant une partie de son diagramme privé ou en ajoutant de nouveaux éléments au diagramme de groupe. Les apprenants expriment leurs opinions dans la fenêtre de chat en utilisant des catégories de communication prédéfinies (*request, inform, motivate, task, maintain, argue, acknowledge, introduce&plan* et *disagree* [Baghaei 2007]).

L'interaction est conçue sous forme de sessions dans lesquelles les apprenants résolvent tout d'abord le problème individuellement et rejoignent ensuite un petit groupe pour créer une solution de groupe. Collect-UML fournit des rétroactions lors des deux phases : durant la phase individuelle, il fournit des rétroactions sur chaque solution, alors que pendant la phase de groupe, il commente la solution de groupe en la comparant à toutes les solutions des membres du groupe. Parallèlement à cela, il fournit des rétroactions relatives à la collaboration issues d'un modèle de collaboration exprimé sous forme de vingt-cinq méta-contraintes représentant une collaboration idéale. À chaque méta-contrainte est associé un message de rétroaction fourni lorsqu'elle est enfreinte (comme pour les contraintes du niveau domaine). Ces méta-contraintes sont divisées en quatre groupes permettant d'observer :

- les contributions des étudiants dans le diagramme de groupe ;
- les contributions dans la fenêtre de chat et l'utilisation des catégories de communication ;
- les différences entre la solution individuelle et la solution de groupe ;
- la planification initiale d'attaque du problème.

Le système utilise une architecture distribuée, où la fonctionnalité de tutorat est distribuée entre le client et le serveur. Le serveur possède un dispositif de modélisation de l'apprenant, qui crée et maintient les modèles de l'apprenant de tous les utilisateurs. Ce serveur comporte également un module domaine, un module pédagogique et un dispositif de modélisation du groupe. Dans cette configuration, un seul apprenant peut mettre à jour le diagramme partagé à un instant donné mais un conseil peut être donné à un apprenant en comparant son travail avec une solution de l'expert ou avec une solution de groupe.

## 2.4 Diagram

L'environnement Diagram est un EIAH conçu dans le cadre du projet I&C (**I**nteraction et **C**onnaissance) du LIUM (**L**aboratoire d'**I**nformatique de l'**U**niversité du **M**aine). Ce projet vise à élaborer des modèles, méthodes et outils pour la conception d'environnements dédiés à l'apprentissage particulier de la modélisation, en se fondant sur le postulat selon lequel la connaissance émerge de l'interaction. Ce projet est appliqué dans Diagram à l'apprentissage individuel des concepts de la modélisation orientée objet par des étudiants novices. Le formalisme UML (cf. chapitre 1.3) comprend plusieurs types de diagrammes, dont le diagramme de classes UML qui est le plus employé et le mieux connu. Diagram se limite actuellement à l'initiation de la construction de diagrammes de classes UML de niveau analyse[10]. Il est admis que le public cible a une connaissance théorique de la syntaxe et de la sémantique du langage UML, et des connaissances générales de la langue naturelle (français) et du monde réel.

Nous définissons dans la suite de cette partie tout d'abord le cadre théorique, puis le modèle d'interaction adopté et enfin le rôle effectif de l'enseignant dans Diagram. Nous présentons seulement ici les aspects de Diagram avant l'intégration de nos travaux sur le diagnostic des productions de l'apprenant.

### *2.4.1 Cadre théorique*

La tâche de modélisation (à la fois celle qui met en jeu la compétence cible, et celle qui est prescrite aux apprenants dans l'enseignement) est une activité de conversion d'un registre langagier à un autre symbolique, avec des actions de manipulation et de transformation sur les représentations elles-mêmes. La nature de cette tâche (**tâche ouverte**), et l'absence de méthodes formelles pour construire un diagramme ou vérifier son adéquation à l'énoncé, nécessitent de mettre l'accent sur l'acquisition par l'apprenant de procédures de contrôle, de correction et de validation de ses productions, procédures qui relèvent du niveau métacognitif.

La notion de métacognition proposée par Flavell se décline en deux dimensions [Flavell 1977] : les connaissances métacognitives d'un individu et les régulations métacognitives qui se réfèrent aux activités supportant le contrôle individuel de la pensée ou de l'apprentissage. Brown distingue trois fonctions de régulation [Brown 1987] : la **planification d'activités** (*planning*), le **contrôle et la surveillance d'activités** (*monitoring*) et la **vérification des résultats de l'activité** (*checking*). Nous faisons l'hypothèse qu'un EIAH supportant une tâche dans laquelle l'activité métacognitive est essentielle doit prendre en compte ces trois dimensions.

L'environnement Diagram est conçu de manière à conduire l'apprenant, au travers de l'interaction, à mobiliser les trois fonctions de la régulation métacognitive et ainsi à faciliter l'acquisition des concepts de la modélisation orientée objet en suscitant l'émergence de schèmes d'action instrumentée pour réaliser efficacement la tâche prescrite. L'environnement doit pour cela favoriser et encourager **l'activité réflexive et métacognitive** de l'apprenant par rapport à son travail, en lui offrant des moyens explicites d'analyse et de justification à l'interface. Enfin, puisque la validation

[10] Nous considérons que les éléments représentables dans un diagramme de classes UML de niveau analyse sont des classes concrètes ou abstraites (décrites par un nom) reliées par des relations d'héritage ou d'association (association, agrégation et composition avec un nom optionnel). Ces classes peuvent encapsuler des attributs et des opérations décrits par au moins un nom. Enfin des multiplicités et des rôles peuvent être reportés aux extrémités de certaines relations d'association.

d'un diagramme n'est pas automatisable, l'interaction est centrée sur la négociation et la co-validation des productions entre l'apprenant et l'EIAH.

### 2.4.2 Modèle d'interaction

L'environnement Diagram comporte un sous-ensemble des fonctionnalités des éditeurs UML classiques. Il propose uniquement les éléments graphiques nécessaires à l'élaboration d'un diagramme de classes UML et simplifie l'édition des différentes caractéristiques des éléments. De plus, Diagram offre la possibilité de travailler simultanément avec l'énoncé (décrivant les spécifications de l'exercice à modéliser) et avec le diagramme de classes UML, ce qui facilite le contrôle visuel de la modélisation. Cette fonctionnalité donne de plus grandes possibilités d'interaction car l'apprenant peut sélectionner des éléments de l'énoncé et modifier son aspect visuel.

Diagram propose trois types de scénarios de modélisation : le premier consiste à construire un diagramme complet à partir d'un énoncé (c'est cette activité qui nous intéresse plus particulièrement). Les deux autres scénarios consistent à compléter un diagramme partiel et à corriger un diagramme erroné. Dans ces deux scénarios, un diagramme est déjà représenté à l'interface au démarrage de Diagram alors que dans le scénario de construction complète d'un diagramme, seul l'énoncé est fourni à l'apprenant.

Dans le cadre de la construction d'un diagramme complet, Diagram intègre une organisation de la tâche qui suit une méthode préconisée par un expert pédagogue du domaine d'apprentissage de la modélisation orientée objet (un des participants du projet I&C). Cet expert préconise d'accomplir en trois phases la tâche de modélisation à partir d'un énoncé textuel :

- Une **première lecture de l'énoncé**, au cours de laquelle le sujet élabore une représentation mentale des objets décrits dans le texte ;
- Une **phase de construction du diagramme** consistant à transposer chaque phrase, sans respecter nécessairement la linéarité du texte, dans un processus itératif qui peut nécessiter des retours-arrières et des corrections ;
- Une dernière **étape de contrôle de la validité du diagramme** par relecture du texte et vérification de l'adéquation du modèle à l'énoncé.

La division en trois étapes a pour objectif de favoriser chez l'apprenant principalement les fonctions de régulation métacognitive de *monitoring* et de *checking* notamment en l'encourageant à planifier sa tâche et à contrôler lui-même la validité de son modèle construit. Ces trois étapes sont réifiées à l'interface de Diagram et ce dernier offre des aides contextuelles qui encouragent l'activité métacognitive chez l'étudiant.

Durant la première étape de lecture (cf. figure 24), l'étudiant découvre l'énoncé. Sur le papier, certains étudiants utilisent un crayon ou un surligneur pour marquer les mots qui leur semblent pertinents pour la modélisation. Pour qu'ils retrouvent cette fonctionnalité dans l'EIAH, la possibilité de souligner des mots de l'énoncé a été intégrée dans Diagram. Une contrainte est imposée avant le passage à la deuxième étape : l'étudiant doit avoir souligné les concepts importants de l'énoncé (définis par l'enseignant lorsqu'il crée l'exercice) avant de passer à la seconde étape.


**Figure 24 : Interface graphique de l'étape 1 de Diagram**

La seconde étape (cf. figure 25) consiste à créer le diagramme de classes correspondant à l'énoncé donné. Diagram permet de créer un élément UML (classe, attribut, opération ou relation) de deux manières : directement à partir d'une expression de l'énoncé (« mode assisté » de création) et librement (« mode libre » de création). En mode assisté, l'élément graphique UML modélisé et l'expression sélectionnée dans l'énoncé sont affichés dans une couleur identique, afin de faciliter le contrôle visuel. Cette modalité de création vise à renforcer le lien entre énoncé et modèle. En mode libre, l'apprenant peut créer librement un élément du diagramme de classes en cliquant sur le bouton puis sur le type de l'élément qu'il désire ajouter au diagramme (cet élément apparaît en noir dans le diagramme). Cette modalité permet de modéliser des éléments qui ne sont pas décrits explicitement dans l'énoncé ainsi que des éléments n'ayant pas de nom (les relations d'héritage par exemple). L'étudiant a ensuite la possibilité de lier entre elles les expressions représentant un même concept dans l'énoncé en sélectionnant une expression et en la rattachant directement à un élément graphique existant ou une autre expression à partir du menu contextuel. Les expressions rattachées prendront la même couleur que l'élément sélectionné pour faciliter le contrôle visuel. L'étudiant est libre également de modifier les noms des éléments créés tout au long de l'activité de modélisation.

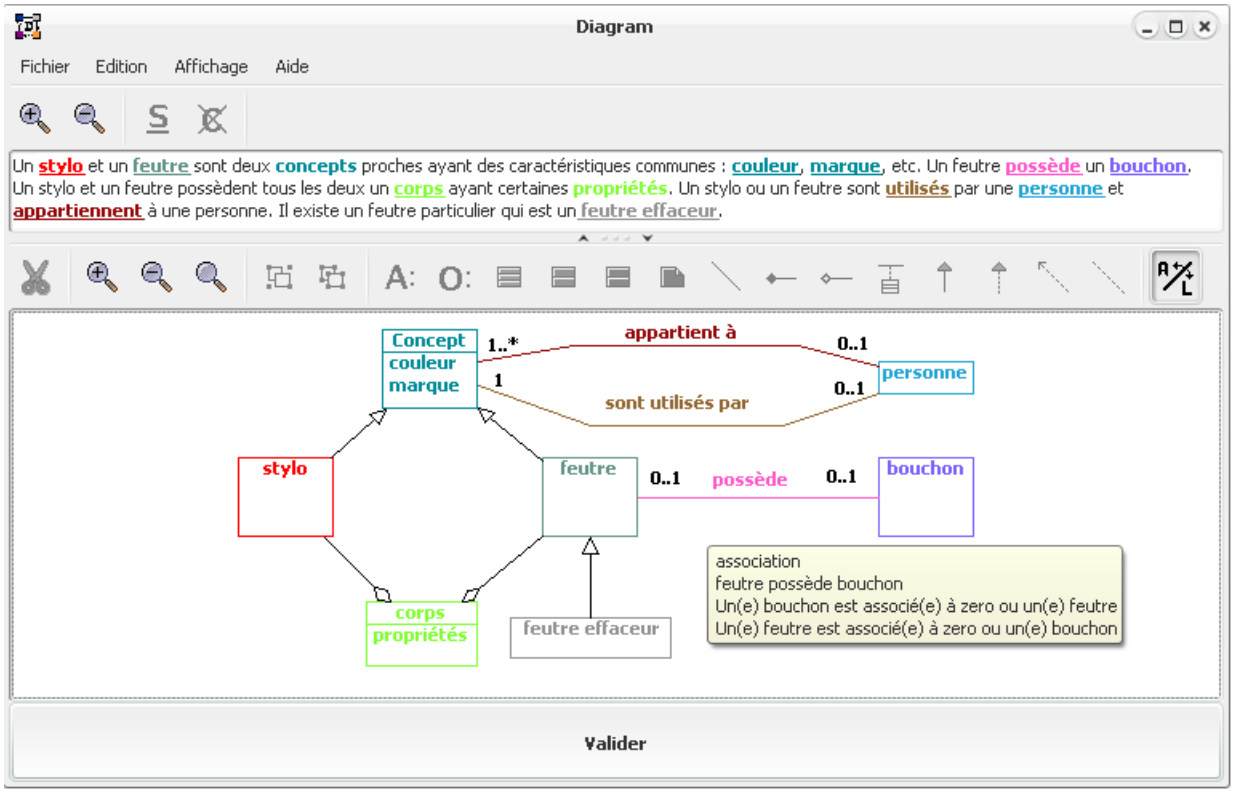



**Figure 25 : Interface graphique de l'étape 2 de Diagram**

L'étape de construction du modèle intègre différentes aides contextuelles qui favorisent le contrôle de l'activité. Par exemple, l'interface offre un dispositif de reformulation textuelle des éléments du diagramme. Lorsque la souris pointe sur un élément du diagramme (classe ou relation), un message indiquant la sémantique de cet élément est affiché dans une infobulle (cf. encadré jaune de la figure 25). Des expérimentations ont montré que ce dispositif favorise l'auto-correction chez l'utilisateur [Alonso *et al.* 2008].

Durant la dernière étape, l'apprenant doit relire l'énoncé de l'exercice et le comparer à son diagramme de classes afin de vérifier que le modèle est complet et correct par rapport à l'énoncé. Au début de cette phase, l'interface ne présente plus que l'énoncé, en « noir et blanc ». Au fur et à mesure du passage de la souris sur l'énoncé par l'étudiant, les expressions modélisées redeviennent en couleur ; simultanément, les éléments correspondants dans le diagramme réapparaissent à l'interface.

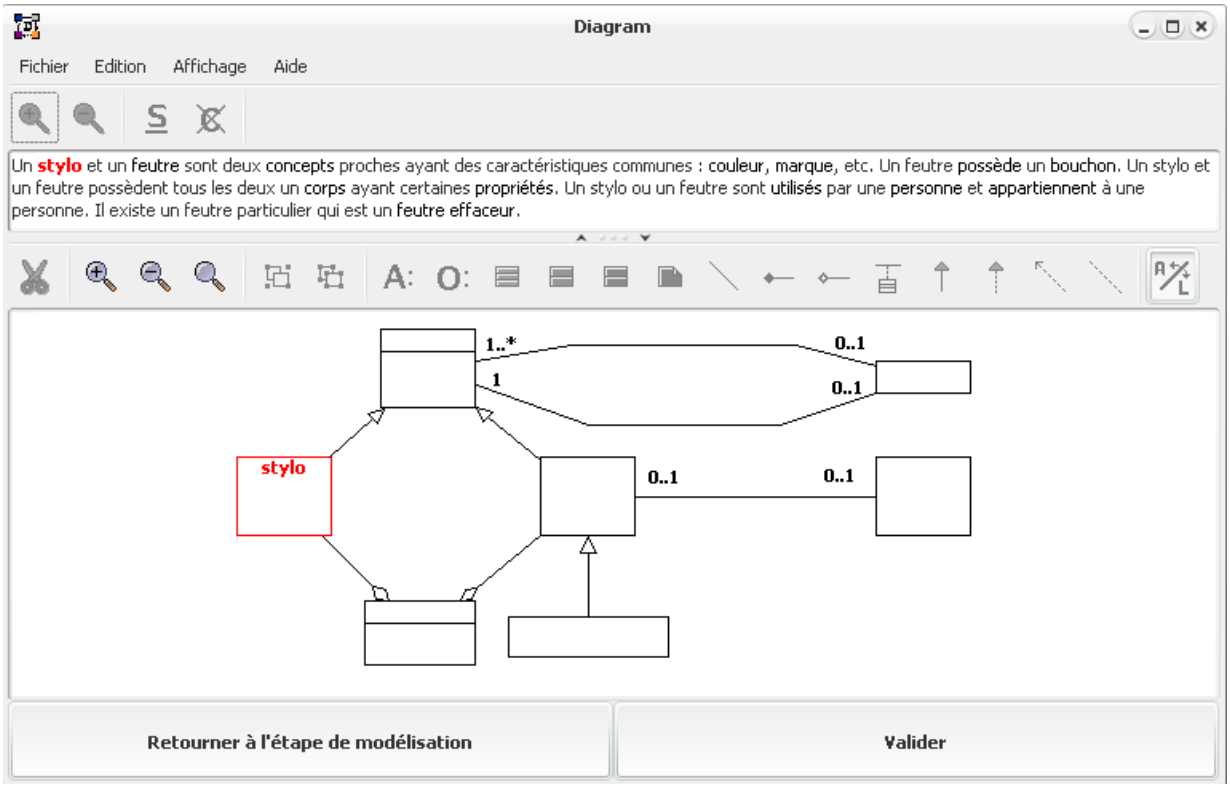



**Figure 26 : Interface graphique de l'étape 3 de Diagram (sans affichage des rétroactions pédagogiques)**

Les aides prévues dans le cadre général d'interaction décrit précédemment sont génériques : elles ne tiennent pas compte de la validité du diagramme de l'apprenant. L'intégration d'un outil de diagnostic dans Diagram pour permettre l'élaboration de rétroactions plus spécifiques, basées sur l'analyse des productions individuelles de l'étudiant, sera décrite dans la suite du mémoire.

### 2.4.3 Rôle de l'enseignant

L'environnement ne corrige pas les erreurs commises par l'apprenant et n'a pas pour objectif de se substituer à l'enseignant au cours de l'activité de construction d'un diagramme UML. Diagram assiste l'apprenant dans son travail en encourageant notamment l'auto-correction. L'enseignant reste présent lors de l'activité de modélisation (menées lors de séances de travaux pratiques) pour prodiguer des conseils à l'apprenant.

Préalablement à l'activité de modélisation de l'apprenant, l'enseignant détermine le contenu de chaque exercice dans une interface spécifique de Diagram. Pour cela, dans l'interface « enseignant » de Diagram, l'enseignant rédige tout d'abord l'énoncé de l'exercice puis il surligne les expressions représentant les concepts importants de l'énoncé qui servent de contraintes au passage de la première étape à la seconde étape de modélisation pour l'apprenant. Enfin, il construit directement un diagramme de classes UML représentant une solution valide (de la même manière que l'apprenant le fait dans la seconde étape de la figure 25). Lors de l'élaboration de l'exercice, l'énoncé et le diagramme de référence définis par l'enseignant sont libres : il n'y a pas d'interdiction dans Diagram lors de l'élaboration de l'exercice. Mais, étant donné que les novices sont le public cible, il est opportun pour l'enseignant de privilégier des énoncés non ambigus, des solutions « naturelles » pour les exercices proposés. Ces recommandations sont identiques à celles qui s'appliquent de manière générale à la conception d'exercices destinés à des novices dans tous les domaines.

# CHAPITRE 3

# L’analyse automatique des productions de l’apprenant dans les EIAH

# PLAN DU CHAPITRE

Nous nous concentrons dans ce chapitre sur l'analyse des réponses de l'apprenant dans le cadre des EIAH proposant des problèmes notamment dans le domaine scientifique (mathématiques, physique, informatique...). Dans ce contexte, un problème, généralement sous la forme d'un énoncé, est proposé à l'apprenant qui, en utilisant ses connaissances personnelles, va exprimer une réponse au problème, qui constitue sa solution (son point de vue). La réponse de l'apprenant dans ce contexte est constituée à la fois des actions que l'apprenant réalise dans l'environnement et de ses productions vues comme les résultats des actions.

La résolution de problèmes exige de l'apprenant une attitude stratégique nécessitant une utilisation flexible et précise de ses connaissances personnelles [Lucangeli & Cornoldi 2001]. Pour résoudre un problème ou réaliser une tâche, trois types de connaissances peuvent être activées [Vivet & Baron 1987] [Giasson 2001] :

- Des **connaissances déclaratives** : ces connaissances correspondent aux **savoirs** d'un sujet sur un objet, une notion ou encore une stratégie de résolution de problèmes, etc. Ces connaissances répondent à la question **quoi**. Connaître les propriétés d'un objet, la définition d'un concept ou savoir que telle stratégie de résolution de problèmes est efficace sont quelques exemples de connaissances déclaratives.
- Des **connaissances procédurales** : ces connaissances correspondent aux **savoir-faire** d'un sujet sur la façon d'employer un objet, une notion ou encore une stratégie de résolution de problèmes, etc. Ces connaissances répondent à la question **comment**. Par exemple, savoir utiliser un appareil, un concept ou une stratégie donnée sont des connaissances procédurales.
- Des **connaissances stratégiques (conditionnelles)** : ces connaissances correspondent aux savoirs d'un sujet sur les conditions d'emploi ainsi que les raisons qui font que l'usage d'un objet, d'une notion ou encore d'une stratégie de résolution de problèmes sont importantes dans telle ou telle situation. Ces connaissances sont des méta-connaissances ou des heuristiques permettant de répondre aux questions **pourquoi** et **quand**. Savoir à quoi sert un outil, savoir dans quel contexte utiliser un concept ou savoir à quel moment utiliser une stratégie sont des exemples de connaissances stratégiques.

Lors de l'activité de résolution, l'apprenant n'a pas nécessairement conscience des connaissances qu'il utilise et le système n'a pas explicitement la possibilité d'y accéder. Les connaissances de l'apprenant mises en jeu lors de l'activité de résolution de problèmes se sont pas accessibles pour l'environnement mais peuvent se manifester par ses actions et ses résultats. L'analyse des réponses d'un apprenant est ainsi un problème central dans le cadre des EIAH et plus particulièrement des STI (**S**ystèmes **T**utoriels **I**ntelligents) [Nicaud & Vivet 1988] car elle peut être utilisée pour essayer de comprendre l'apprenant en inférant la représentation qu'il a du problème et les connaissances qu'il utilise, de découvrir ses difficultés et de l'assister au cours de l'activité. Pour analyser ce que fait l'apprenant, quand il le fait et la manière dont il le fait, le système se base sur les **observables** : ce sont usuellement les actions effectuées à l'interface (par exemple les clics de souris, les saisies de texte) recueillies par le système lors d'une session de son utilisation par l'apprenant. La nature des observables est liée au domaine d'application, et leur granularité dépend de l'environnement. À partir de ces observables, plusieurs questions se posent :

*Comment analyser et interpréter les réponses de l'apprenant ?*

*Quelle exploitation pédagogique peut-on faire du résultat de l'analyse des réponses ?*

Nous apportons quelques réponses à ces questions en présentant tout d'abord les objectifs généraux et les méthodes permettant d'analyser les réponses de l'apprenant dans les EIAH. Nous nous focalisons ensuite sur les spécificités et les méthodes d'analyse propres à la résolution de problèmes. Nous exposons enfin en détail les méthodes d'analyse dans le cadre de l'apprentissage de la modélisation dans lequel nos travaux s'inscrivent.

## 3.1 Objectifs et méthodes

Un système d'EIAH est classiquement découpé en quatre modèles [Wenger 1987] :

- Le **modèle du domaine** contient les connaissances nécessaires pour résoudre les problèmes proposés à l'apprenant ;
- Le **modèle pédagogique** contient les connaissances nécessaires à la remédiation et au dialogue tutoriel ;
- Le **modèle de l'interaction** contient les connaissances nécessaires pour mener à bien l'interaction entre l'apprenant et le système ;
- Le **modèle de l'élève** contient les connaissances censées expliquer le comportement de l'apprenant.

L'analyse automatique des productions des apprenants à partir des observables peut avoir des objectifs multiples dans les EIAH. Nous exposons ici une liste non exhaustive des objectifs possibles :

- **L'évaluation** de la réponse : elle peut être réalisée selon plusieurs critères et consiste à déterminer si la réponse ou la proposition de l'apprenant est correcte ou non (pour une connaissance élémentaire) et éventuellement à identifier si la stratégie employée par l'apprenant est optimale ou non. L'apprenant peut avoir commis des erreurs ou manifester des conceptions incorrectes (*misconceptions)* lors de la résolution de la tâche.
- **La production de rétroactions pédagogiques** : l'analyse des réponses de l'apprenant peut servir à construire des rétroactions guidant l'apprenant ou apportant des conseils adaptés lors de l'activité d'apprentissage.
- La **modélisation de l'apprenant** : elle consiste à inférer les connaissances de l'élève à partir de ses réponses en vue de construire un modèle de l'élève. Dans un STI, le **modèle de l'élève** est une représentation des connaissances de l'apprenant, qu'elle soit qualitative ou quantitative, formelle ou informelle, et qui rend compte complètement ou partiellement d'aspects spécifiques du comportement de l'élève [Sison & Shimura 1998].
- L'adaptation des contenus de formation en fonction des connaissances de l'apprenant.
- La constitution de groupes d'apprenants.

Pour analyser les réponses de l'apprenant, le système utilise des connaissances propres. Les méthodes employées ont recours en général à un modèle de l'expertise capable de **résoudre le problème** ou de **valider un pas de raisonnement ou une réponse**. Les méthodes utilisant un résolveur peuvent déterminer la ou les réponses possibles à un problème posé puis comparent la réponse de l'élève avec les réponses possibles. Il suffit alors de procéder par appariement pour déterminer si la réponse est correcte ou non. Cette méthode est applicable notamment dans des domaines fermés ou lorsque la tâche à réaliser est simple. Lorsque toutes les réponses ou que tous les raisonnements ne sont pas calculables, le système peut valider une réponse en reconstruisant le raisonnement effectué à partir de la réponse de l'apprenant. GUIDON [Clancey 1983], dans le domaine du diagnostic d'infections bactériennes en

médecine, repose sur cette méthode : il commence par résoudre un cas de diagnostic à l'aide du système expert MYCIN puis propose le même cas à l'étudiant qui propose ses diagnostics et pose des questions lorsque des compléments d'information lui sont nécessaires. Pour évaluer les réponses de l'étudiant, GUIDON se base sur le schéma suivi par un expert pour aboutir à la réponse. Ce schéma est compilé par MYCIN sous la forme d'une arborescence *et/ou* incluant une mention explicite sur chaque résultat intermédiaire, qu'il soit juste ou faux. Pour cela, MYCIN raisonne sur la base d'hypothèses constituées de règles où chaque règle aboutit à un résultat intermédiaire qui va définir l'hypothèse suivante et ce successivement jusqu'à obtenir la bonne réponse. GUIDON considère que si l'apprenant applique les bonnes règles, il doit arriver aux mêmes conclusions que le système. GUIDON représente les connaissances de l'apprenant par un jeu de trois valeurs enchâssées (croyance dans la connaissance de la règle, croyance dans la capacité à l'appliquer et croyance dans l'application effective dans une situation donnée).

Les méthodes d'analyse des productions de l'apprenant considèrent de différentes manières la notion d'erreurs et de *misconceptions*. L'erreur peut être **implicite** : dans ce cas, on considère qu'elle provient d'une absence de connaissance chez l'apprenant. Généralement, une représentation des bonnes connaissances ou des réponses de l'expert est à disposition des systèmes qui considèrent les erreurs de cette manière. Les tuteurs WEST [Burton & Brown 1982] et WUSOR [Goldstein 1982] considèrent que c'est l'ignorance de telle règle ou de tel concept qui amène l'apprenant à ne pas faire le meilleur choix possible. L'inconvénient majeur ici est de supposer que si l'apprenant n'utilise pas une connaissance de l'expert pour construire sa solution, alors c'est que l'apprenant ne maîtrise pas cette connaissance. On ne prend pas en compte la possibilité de stratégies alternatives correctes, qui ne seraient pas modélisées par le système. L'erreur peut être **explicite** : dans ce cas elle est définie sous forme de bibliothèques de bogues (des règles ou des stratégies erronées). Les systèmes les plus représentatifs de cette approche sont BUGGY [Brown & Buron 1978] et ses successeurs DEBUGGY [Burton 1982] et IDEBUGGY dont les hypothèses de base sont d'une part que les savoir-faire en arithmétique (ici la soustraction) peuvent être représentés par des procédures organisées en réseau et d'autre part, que les erreurs ont un caractère systématique et s'expliquent par la présence de perturbations dans les procédures. Ces environnements ont démontré à la fois la faisabilité et la pertinence, pour un domaine restreint donné, de la mise au point d'un catalogue complet d'erreurs de nature procédurale.

## 3.2 Méthodes d'analyse en résolution de problèmes

Le domaine de la résolution de problèmes possède des spécificités propres. Les connaissances entrant en jeu lors de la résolution du problème ne sont pas uniquement déclaratives (**savoirs**) mais également procédurales (**savoir-faire**) et stratégiques (**quand et pourquoi utiliser les savoirs et savoir-faire**). Les connaissances procédurales et stratégiques peuvent notamment être exprimées sous la forme d'heuristiques et de règles dans des systèmes experts. L'apprenant, pour résoudre le problème, va effectuer un raisonnement en plusieurs étapes (i.e. des pas de raisonnement) en suivant une stratégie donnée. Les activités métacognitives de planification (i.e. imaginer comment procéder pour résoudre un problème), de monitorage (i.e. évaluer l'efficacité d'une stratégie), et de régulation (i.e. ajuster l'efficacité de la stratégie) sont très importantes lors de la résolution d'un problème.

### 3.2.1 Approche de type « guidage discret »

Les principes généraux de cette approche sont de modéliser les connaissances procédurales et stratégiques dans un modèle d'expertise de type « boîte noire » et de se focaliser sur la stratégie suivie par l'apprenant. L'analyse des réponses de l'apprenant est réalisée à chaque étape de la résolution du problème. En général, l'expert produit l'ensemble des réponses possibles et étiquette chaque réponse en fonction de la stratégie appliquée. La réponse de l'apprenant est ensuite appariée avec l'une de ces réponses et la stratégie suivie peut ainsi être déduite. Si la stratégie n'est pas optimale, l'environnement intervient et présente à l'apprenant la réponse qui aurait pu être fournie en appliquant une stratégie optimale. Le système critique donc les réponses de l'apprenant.

Le « guidage discret » est un mode d'interaction consistant à laisser l'apprenant agir, observer ce qu'il fait et n'intervenir qu'à certains moments biens choisis (généralement lorsque l'apprenant a commis plusieurs fois la même faute) pour enseigner l'heuristique pertinente. Le principe de « guidage discret » a été mis en place dans les systèmes WEST, WUSOR et GUIDON.

Le système WEST développé par Burton et Brown [Burton & Brown 82] est un environnement de jeu de type micromonde inspiré du « jeu de l'oie » qui propose une course entre l'ordinateur et l'élève pour la conquête de l'Ouest. Le jeu comporte soixante-dix cases dont certaines correspondent à des villes. Le but du jeu est d'atteindre le premier la dernière ville. Ce jeu a deux intérêts éducatifs : d'une part l'enseignement des opérations arithmétiques et d'autre part l'enseignement des heuristiques du jeu. Le jeu consiste, pour chaque joueur, à combiner trois nombres tirés au hasard par la machine avec des opérateurs arithmétiques de son choix. Le résultat de cette combinaison détermine le nombre de cases (le score) dont il avance ou recule sur le chemin (le mouvement du joueur). Le jeu comporte des raccourcis de plusieurs types qui font que la stratégie optimale de jeu n'est pas forcément celle du calcul du plus grand score. Notamment, quand un joueur atteint la case de son adversaire, ce dernier recule de façon importante (vers la deuxième ville précédente) et lorsqu'il atteint une ville, cela lui permet d'aller directement à la suivante. À chaque fois que l'élève joue, son mouvement est comparé à un ensemble de mouvements alternatifs construits par le module expert dans la même situation que l'élève (la stratégie par défaut dans WEST est de maximiser la différence de cases avec l'opposant). La différence entre la solution de l'élève (l'expression mathématique) et les bonnes solutions de l'expert permet de fournir des hypothèses sur ce que l'élève ne connaît pas ou n'a pas encore maîtrisé. À partir de cette analyse et de principes généraux de guidage (cf. figure 27), le système décide ou non d'interrompre le déroulement de la partie.

Le guide mis en place pointe notamment les opportunités (en proposant une stratégie optimale à l'apprenant) pour faire prendre conscience à l'apprenant des possibilités non encore explorées du jeu. Le système veille à transformer les échecs en expériences d'apprentissage sans pour autant empêcher l'apprenant de se tromper. Un inconvénient majeur de WEST est que le système suppose que si l'élève n'utilise pas une connaissance de l'expert pour construire sa solution, alors l'élève ne maîtrise pas cette connaissance.

Guider de manière discrète

1. Avant de donner un conseil sur une méthode, s'assurer que le joueur la maîtrise mal.
2. Pour illustrer une méthode, utiliser uniquement un exemple pour lequel le résultat est largement supérieur à celui du joueur.
3. Après lui avoir donné un conseil, permettre au joueur de l'appliquer immédiatement en l'autorisant à rejouer.
4. Si le joueur est en train de perdre, ne l'interrompre que si l'on peut lui éviter de perdre.

Maintenir l'intérêt du jeu

5. Ne pas intervenir deux fois de suite, quelle qu'en soit la raison.
6. Ne pas intervenir avant que le joueur n'ait eu l'occasion de découvrir le jeu par lui-même.
7. Ne pas donner uniquement des critiques. Féliciter l'élève pour ses choix judicieux.
8. Après avoir donné un conseil, lui permettre de rejouer mais ne pas l'imposer.

Multiplier les occasions d'apprentissage

9. Faire en sorte que l'ordinateur joue toujours de manière optimale.
10. Si l'élève demande de l'aide, lui fournir plusieurs niveaux de conseil.

Tenir compte de l'environnement ludique

11. Si le joueur perd largement, ajuster le niveau de jeu[8].
12. Si l'étudiant a pu faire une erreur d'inattention, être magnanime mais afficher un message explicite pour le cas où cela n'aurait pas été de l'inattention.

**Figure 27 : Les 12 principes de guidage discret de WEST [Burton & Brown 1982] dans [Bruillard 1997]**

Le système WUSOR [Goldstein 1982] est un tuteur sur un jeu inspiré de « Thésée et le Minotaure » tiré de la mythologie grecque. Le joueur doit passer par des cases successives où un monstre peut se cacher. D'autres dangers peuvent être présents sur les cases telles que des chauves-souris ou des fosses. Pour choisir la case suivante, l'élève utilise la logique et les probabilités pour éviter les dangers. Les connaissances de l'élève sont représentées comme un sous-ensemble des connaissances de l'expert et sont structurées dans un **graphe génétique**. Chaque nœud du graphe est une procédure et les arcs représentent les relations entre les procédures. Les connaissances semblables sont donc représentées par des nœuds voisins dans le graphe. Goldstein souligne le fait que les connaissances nouvellement acquises sont proches des connaissances déjà acquises, et focalisent le diagnostic à la frontière du modèle de l'élève. Les interventions pédagogiques de WUSOR sont fournies sous forme de conseils portant sur les connaissances proches de celles déjà acquises.

Le « guidage discret » est bien adapté à l'enseignement des heuristiques (plus spécialement celle acquises par l'observation et l'expérience) car les stratégies y sont explicites dans le modèle. L'environnement d'apprentissage peut donc baser son analyse des réponses et ses explications sur ces connaissances. Néanmoins, cette approche ne peut s'appliquer qu'à des domaines très simples où les heuristiques sont bien définies et exprimées en faible nombre. Le problème général de la représentation des connaissances de l'élève comme découlant directement de la représentation des connaissances dans le module expert est une limite car les connaissances du novice correspondent bien souvent non pas à un sous-ensemble des connaissances de l'expert mais à une conceptualisation globalement différente du domaine. La seconde limite est liée au modèle qui considère qu'une performance non optimale est le résultat d'un manque de connaissance par rapport à celle de l'expert et non celui d'une erreur commise lors du déroulement d'une stratégie de solution.

### 3.2.2 Approches en résolution de problèmes

Dans le contexte de la résolution de problèmes du domaine scientifique (en mathématiques, physiques ou en informatique), il est nécessaire habituellement de représenter les connaissances procédurales dans le modèle d'expertise avec des variantes erronées pour pouvoir analyser les réponses de l'apprenant. Il n'y a pas nécessairement de représentation explicite des stratégies mais on considère communément que les solutions comportant un petit nombre d'étapes sont préférables. Le principe souvent retenu est qu'un module expert produit avant la session l'ensemble des réponses correctes possibles constituées de plusieurs pas de raisonnement. Chaque réponse de l'apprenant est appariée avec une de ces étapes pour chercher à en déduire progressivement le raisonnement suivi par l'apprenant. L'EIAH peut ainsi proposer des aides à l'apprenant en lui donnant un indice sur la prochaine étape à réaliser. Cette analyse des réponses de l'apprenant a notamment été traitée de manière numérique dans Andes et symbolique dans Mentoniezh. D'autres systèmes, notamment ceux utlisant le *model tracing* produisent à l'avance des solutions erronées en plus des solutions correctes. Enfin, certains systèmes tels que Aplusix ne produisent pas les réponses possibles avant la session.

Dans le domaine de la résolution de problème en physique de niveau universitaire, Andes [Gertner & VanLehn 2000] est un tuteur intelligent proposant à l'étudiant de résoudre des problèmes de mécanique newtonienne. Une description textuelle du problème et une image représentant la situation du problème sont fournies à l'apprenant. À partir de ceci, l'apprenant peut construire un diagramme librement, entrer les équations utilisées pour résoudre le problème et toutes les variables apparaissant dans les équations. L'apprenant doit en revanche définir les variables à partir de la description textuelle du problème pour qu'Andes soit capable d'interpréter ce qu'elles représentent.

Andes dispose d'une base de connaissances de 540 règles (constituée par des enseignants d'université) permettant de résoudre hors-ligne 120 problèmes de physique. Il utilise ces règles également pour instancier un modèle de l'apprenant sous la forme d'un réseau bayésien [Conati *et al.* 2002]. Les règles peuvent représenter les concepts du domaine de la physique ou guider le système pour obtenir les étapes d'une solution (*goal rules*). Pour chaque problème, un résolveur automatique utilisant ces règles génère et sauvegarde le ou les chemins de solution dans un réseau spécifique, nommé **graphe solution**. Ce graphe est constitué de toutes les applications des règles relatives au problème spécifique ; il peut inclure des nœuds pour plus d'une solution acceptable et des nœuds erronés (*buggy nodes*) représentant des *misconceptions* ou des erreurs courantes. Un algorithme interprète ce que fait l'apprenant au fur et à mesure de son interaction avec le système, en mettant à jour les poids des nœuds du réseau, qui modélisent sous forme de probabilités les multiples règles possibles pour expliquer ses actions. Le graphe solution est utilisé pour déterminer l'étape atteinte dans la résolution par l'apprenant et pour identifier quelle pourrait être sa prochaine étape. Pour déterminer si un pas de raisonnement entré par l'apprenant est correct, Andes l'apparie donc à un nœud du graphe solution. Si le pas de l'apprenant s'apparie à un nœud erroné ou ne correspond à aucun nœud, Andes l'étiquette comme une erreur. Andes trouve également dans ce cas le nœud qui est le plus proche d'une action de l'apprenant et utilise cette information pour qualifier l'erreur dans cette étape. Il fournit ensuite des conseils sur cette erreur. Lorsque l'apprenant décrit sa solution, les informations correctes à l'interface sont colorées en vert et celles incorrectes sont mises en évidence en rouge. L'étudiant peut demander de l'aide pour corriger une erreur et le système adapte les informations à chaque demande pour l'encourager à corriger ses erreurs lui-même : il fournit tout d'abord un conseil succinct, puis un conseil spécifique et enfin la démarche à suivre pour corriger l'erreur.

Le tuteur intelligent Mentoniezh [Py 1996] [Py 2001], dans le domaine de la démonstration en géométrie élémentaire, est basé sur un démonstrateur automatique qui produit, pour un problème donné, un ensemble de démonstrations qui constituent les différents plans de résolution. Chaque démonstration est une séquence de pas de preuves qui permet de démontrer la conclusion du problème à partir de ces données. Au cours de la résolution du problème, Mentoniezh utilise la reconnaissance de plans [Kautz 1987] à chaque action de l'apprenant (chaque pas de démonstration) pour trouver les plans qui expliquent au mieux les actions observées. Partant d'une description incomplète des actions effectuées par l'élève, la reconnaissance de plans s'attache à retrouver le but recherché par l'apprenant, ainsi que le ou les plans (la démonstration) parmi l'ensemble des possibles que l'apprenant est en train de construire. L'apprenant peut mener plusieurs tâches de front et Mentoniezh est capable de gérer les problèmes de changement de plans grâce à deux critères de préférence (i.e. des heuristiques) sur le **plan principal** (i.e. le plan regroupant le plus grand nombre d'observations) et les **plans secondaires** (i.e. les autres plans possibles). Mentoniezh propose des explications à l'élève sur ses erreurs. Chaque étape de démonstration est susceptible de donner lieu à des erreurs qui sont définies dans un catalogue d'erreurs réparti en trois classes : **syntaxiques**, **logiques**, **sémantiques** [Py 2001]. Les règles incorrectes potentiellement utilisables par l'élève ne sont pas explicitement décrites dans le système mais sous la forme de méta-règles permettant de construire une règle incorrecte en fonction de l'ensemble des règles correctes. Dans ce système, le guidage prodigué à l'apprenant est souple car une aide n'est donnée qu'à la demande de l'élève. Dans le cas où un plan est reconnu, l'aide consiste à indiquer la propriété suivante à démontrer dans le plan. Dans le cas contraire, l'élève est encouragé à explorer le problème depuis le but, en proposant des décompositions possibles en sous-buts.

La technique de traçage de modèle (*model tracing*) issue des travaux de l'équipe d'Anderson consiste à analyser étape par étape le raisonnement de l'élève à partir de règles de production réparties en deux sous-ensembles (correctes ou erronées) et à intervenir immédiatement lorsque l'élève commet une erreur (la correction différée d'une erreur serait beaucoup plus coûteuse en temps). Le système Lisp Tutor [Anderson & Reiser 1985] est un exemple de tuteur reposant sur cette approche. Il propose un exercice de programmation à l'apprenant qui y répond en tapant son programme. À chaque fois que l'étudiant édite une nouvelle partie de son code, le tuteur tente d'associer cette étape à une des règles de la base. Si la règle associée est incorrecte, le tuteur explique l'erreur. Si l'étudiant ne peut pas continuer en suivant une solution correcte alors le tuteur peut lui montrer l'étape correcte ou faire une démonstration de l'algorithme à programmer. L'avantage de ce type d'approche est de guider systématiquement l'apprenant vers une solution correcte. Cependant, l'apprenant ne peut pas explorer des chemins corrects mais sous-optimaux, et encore moins des chemins incorrects au bout desquels il se rendra compte de ses erreurs par lui-même.

Aplusix [Nicaud 1987] [Bouhineau *et al.* 2003] est un micromonde d'aide à l'apprentissage de l'algèbre formelle avec lequel l'élève développe ses propres calculs. Il est doté pour cela d'un éditeur avancé d'expressions algébriques qui permet leur affichage, leur écriture et leur modification en deux dimensions, en respectant la structure des expressions algébriques. Aplusix porte sur la résolution d'exercices formels (calculs numériques, développements, factorisations, résolutions d'équations…) ainsi que sur la résolution de problèmes. Il contrôle toujours les calculs de l'élève en vérifiant l'équivalence mathématique des expressions entre étapes successives. L'environnement dispose d'une base de règles de réécriture et de production permettant de décrire des transformations algébriquement correctes ou erronées organisées en une taxinomie (conçue par des experts du domaine). Le diagnostic d'une transformation donnée (i.e. tout passage d'une expression à une autre) précise quel type de règle a pu utiliser l'élève lors de telle

résolution d'exercice. Basé sur un mécanisme de recherche heuristique de type « meilleur d'abord » [Pearl 1984], le diagnostic explique la transformation produite par un élève, par une suite d'applications de règles correctes ou erronées de la base de règles construite. Un degré supérieur de modélisation peut être atteint ensuite en regroupant les règles permettant de faire émerger des conceptions. Aplusix propose trois sortes de rétroactions : l'indication de la justesse des calculs à tout moment, la fourniture de la solution à la demande et l'indication de si l'exercice est bien terminé quand l'élève l'affirme. Le diagnostic local dans cette proposition doit être contrôler par des heuristiques et admet des cas d'échecs par exemple lors de l'absence d'une règle dans la base, d'une profondeur trop faible d'exploration.

Les systèmes basés sur ces approches ont l'avantage de pouvoir évaluer l'utilité d'un pas de raisonnement (pas uniquement vrai ou faux). Cependant, les méthodes proposées ne peuvent s'appliquer qu'à des domaines de taille relativement réduite car il est nécessaire de disposer à l'avance de toutes les résolutions possibles. Plus le domaine est vaste, plus il y a de solutions possibles. De plus, les étapes doivent être définies par des règles explicites du fait de l'utilisation de résolveurs. Si certaines règles sont implicites ou absentes, les résolveurs ne sont pas aptes à les prendre en compte et ne peuvent couvrir l'ensemble des résolutions possibles mais seulement un sous-ensemble.

## 3.3 Méthodes d'analyse dans le cadre de l'apprentissage de la modélisation

Le domaine de l'apprentissage de la modélisation apporte des spécificités et des contraintes que les systèmes doivent prendre en compte pour être capables d'analyser les réponses de l'apprenant. La construction d'un modèle à partir d'un énoncé est une tâche ouverte n'imposant pas de méthode ou de stratégie de résolution particulière. Il est difficile ici de déduire directement, à partir des actions de l'apprenant, son raisonnement sous-jacent et les connaissances qu'il emploie. Un problème de modélisation peut susciter chez chaque apprenant une solution différente. Il n'est donc pas envisageable de calculer l'espace des solutions possibles pour ce genre de problème. L'apprenant doit être néanmoins capable de planifier son activité et de contrôler l'adéquation du modèle produit aux spécifications. La tâche de modélisation demande à l'apprenant des connaissances sur le domaine à modéliser, sur le langage naturel, le formalisme de modélisation utilisé et des capacités de planification. Toutes les connaissances ne peuvent être incorporées dans un modèle d'expertise.

Peu de travaux se sont attachés à analyser les réponses de l'apprenant dans le cadre de la modélisation en informatique. Deux approches, l'approche par *Curriculum* et l'approche basée sur les contraintes ressortent de la littérature par leur capacité à identifier les erreurs de l'apprenant, à fournir des rétroactions à l'apprenant en cours d'activité de modélisation. Ces approches ont en commun de recourir à une solution de référence exprimant une représentation correcte d'un problème de modélisation.

### *3.3.1 Approche « Curriculum »*

L'approche que nous nommons *« Curriculum »* est utilisée dans l'environnement DesignFirst-ITS (cf. chapitre 2.2.2) pour l'apprentissage de la modélisation orientée objet. Le terme *« Curriculum »* renvoie au CIN (***C****urriculum* ***I****nformation* ***N****etwork*) liant les différents concepts du domaine d'apprentissage entre eux. Ce CIN, situé au

cœur de l'environnement, est notamment consulté lors de l'analyse des actions de l'apprenant, de la production de rétroactions pédagogiques et de la modélisation de l'apprenant. Un curriculum définit toutes les connaissances devant être acquises lors de l'enseignement par les apprenants, et un CIN décrit les relations entre les concepts et les compétences dans le curriculum. Un CIN est un graphe acyclique orienté dans lequel chaque nœud représente une connaissance distincte, et chaque arc représente une relation de prérequis entre deux nœuds. Un prérequis est un concept que l'apprenant a besoin de connaître avant de pouvoir comprendre un autre concept. Les connaissances que les apprenants ont besoin de posséder pour résoudre des problèmes orientés objet sont modélisées dans le CIN [Wey 2007].

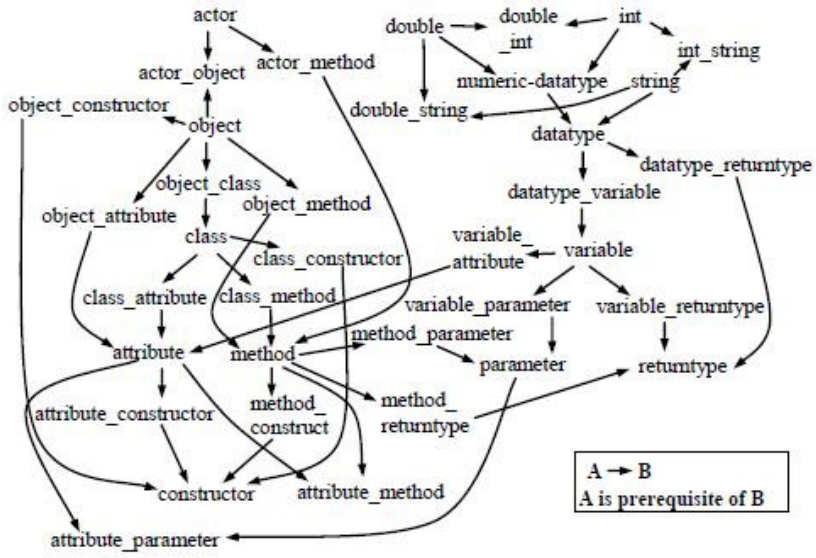



**Figure 28 : Le CIN pour le modèle de l'apprenant [Wey 2007]**

L'analyse des réponses de l'apprenant est réalisée pas à pas par l'évaluateur expert (*Expert Evaluator* dans la figure 21 du chapitre 2.2.2) du module expert de DesignFirst-ITS [Moritz 2008]. Lorsqu'un étudiant entre ou change un élément graphique de son diagramme (classe, attribut ou méthode) dans l'éditeur graphique UML, un enregistrement contenant les détails de l'action est écrit dans la base de données de l'environnement. Cet enregistrement est lu par l'évaluateur expert qui incorpore l'action dans la solution de l'apprenant et qui évalue ensuite si elle est correcte ou non en utilisant un algorithme d'appariement. Le résultat est un « paquet d'informations » pour une action correcte et un « paquet d'erreur » pour une action erronée. Le CIN est consulté enfin pour déterminer les connaissances acquises ou non maîtrisées par l'apprenant : ces dernières sont considérées comme la cause des erreurs qu'il commet dans DesignFirst-ITS.

L'appariement des solutions se déroule ainsi : l'expert tente en premier lieu d'apparier un élément de la solution de l'apprenant à un élément du même type dans la solution de référence. Les éléments qui n'ont pas été appariés précédemment par l'évaluateur expert sont parcourus en premier. Si un appariement est trouvé et qu'il montre que l'apprenant a dupliqué un élément ou supprimé un élément correct du diagramme alors un paquet d'erreur est généré. Dans les autres cas où un appariement est identifié, un paquet d'informations est produit en fonction des résultats. Si aucun appariement n'a pu être trouvé dans cette première étape, le système va comparer le nom de cet élément à ceux des autres types d'éléments dans la solution de référence. Si un élément peut être apparié à ce moment là, un paquet d'erreur est créé. Dans le cas contraire, aucun appariement ne peut être identifié pour l'élément construit par l'apprenant et l'évaluateur ne peut déterminer les intentions de l'apprenant.

L'appariement des éléments se fait essentiellement en prenant en compte les espaces de nommage des éléments UML. L'algorithme permettant de comparer et d'apparier le nom d'un élément, intervenant dans une action de l'apprenant, à un nom de la solution idéale est le même quels que soient les types d'éléments. Les abréviations et les fautes de frappe sont prises en compte en utilisant deux métriques de similarité des chaînes de caractères tirées d'une librairie open-source nommée SimMetrics [SimMetrics]. Un score de similarité est assigné à chaque paire de noms comparés, avec une valeur de 1 pour un appariement exact. Les éléments avec une similarité supérieure à 0,8 sont mémorisés comme des appariements possibles. Si aucun score ne respecte ce critère alors aucun appariement n'est retourné. Si plusieurs candidats sont possibles, les scores sont minorés en fonction de certains types des éléments [Moritz 2008] et l'élément de la solution de référence avec le plus haut score de similarité ainsi calculé est retenu dans l'appariement.

La logique d'appariement précédente s'applique à la fois en cours de modélisation et à la fin de l'activité. Lorsque l'étudiant a terminé la construction de son diagramme, l'évaluateur expert procède de la même manière que lors de l'activité de modélisation : il traite pas à pas les actions qui ont été mémorisées et génère les paquets d'informations et d'erreur correspondant à chacune des actions.

La solution idéale du problème de modélisation est générée automatiquement avant l'activité dans un outil dédié à cet effet nommé *Instructor Tool*. L'enseignant peut y entrer les descriptions textuelles des problèmes, visualiser et réviser les solutions générées. Un générateur de solution traite le texte du problème et génère une solution en utilisant deux modules externes pour analyser les expressions et la structure des énoncés en langage naturel : MontyLingua [MontyLingua] et WordNet [Felbaum 1998] [WordNet]. Cette solution est contrôlée et affinée par l'enseignant. MontyLingua est un processeur de langage naturel utilisé pour analyser les phrases de l'énoncé et en extraire les noms des éléments à représenter dans la solution. Il relève pour cela des triplets verbe/sujet/objet qui sont les bases pour l'identification des classes et de leurs attributs et méthodes associés. MontyLingua a des difficultés pour analyser les phrases complexes, les clauses de dépendances et la reconnaissance des appositions ne sont pas possibles. WordNet est une base de données lexicale en langue anglaise développée à l'université de Princeton qui fournit des services similaires à un dictionnaire et un thesaurus. De plus, WordNet regroupe les noms, les verbes, les adjectifs et les adverbes dans des ensembles de synonymes qui sont liés sémantiquement. Le générateur de solution utilise WordNet pour clarifier le rôle d'un mot dans une phrase et pour identifier des synonymes dans le contexte du problème. Ces synonymes sont utilisés lors de l'analyse de la solution de l'apprenant pour repérer les doublons.

Les auteurs recommandent à l'enseignant qui utilise l'outil de génération de solutions de ne pas introduire de connaissances implicites, de détails superflus, de détails d'implémentation. L'énoncé doit être exprimé de manière simple (i.e. en voix active et avec des phrases simples). Enfin, le diagramme solution ne doit pas représenter plus de cinq classes pour qu'il soit adapté au niveau des apprenants novices. L'enseignant peut indiquer si les éléments de la solution sont requis ou optionnels dans les cas où une représentation alternative est jugée pertinente par l'enseignant. Ces différents choix simplifient l'analyse des actions de l'apprenant car les exercices proposés sont très structurés et ne laissent que très peu de liberté à l'apprenant lors de la construction de sa solution.

### 3.3.2 Approche basée sur les contraintes (CBM)

L'approche CBM (***C**onstraint **B**ased-**M**odelling*), introduite par Ohlsson [Ohlsson 1994], est basée sur la théorie de l'apprentissage à partir des erreurs de performance (*performance errors*) [Ohlsson 1996]. Cette théorie se concentre sur la manière dont les personnes découvrent et corrigent leurs propres erreurs lorsqu'ils exercent leurs compétences. L'hypothèse de base est que les informations nécessaires au diagnostic ne sont pas dans la séquence des actions de l'apprenant, mais dans la situation ou l'état du problème dans lequel l'apprenant arrive. Ohlsson considère qu'une solution correcte ne peut être atteinte en passant par un état du problème qui viole les fondamentaux ou les concepts du domaine. L'analyse des productions et la modélisation de l'apprenant avec CBM ne se fait donc pas sur les actions de l'apprenant mais sur la solution proposée par l'apprenant.

L'approche CBM ne requiert pas l'utilisation d'un module expert exécutable contrairement à de nombreux tuteurs intelligents. C'est un avantage important, car dans beaucoup de domaines (tels que la modélisation), il est très difficile de construire un résolveur de problèmes. Dans l'approche CBM, il n'y a pas non plus de représentation des « connaissances » car seules les contraintes du domaine y sont modélisées : ce sont les règles locales qu'une solution ou qu'un élément de solution doivent respecter. Par exemple, dans une expression algébrique, le fait que le nombre de parenthèses ouvrantes doive être égal au nombre de parenthèses fermantes est une contrainte syntaxique. Dans une addition le fait que la valeur de la somme doive être supérieure à la valeur de chacun des termes sommés est une contrainte sémantique. Les contraintes permettent de rejeter une réponse fausse à coup sûr mais ne garantissent pas qu'une réponse les respectant toutes soit juste.

Dans l'approche CBM, chaque contrainte est une paire ordonnée ($C_r$, $C_s$) où $C_r$ est une condition de pertinence et $C_s$ est une condition de satisfaction. La condition de pertinence décrit les états dans lesquels la contrainte peut être appliquée et la condition de satisfaction définit les états dans lesquels la condition est satisfaite (i.e. la solution est correcte localement à la contrainte). Une contrainte est ainsi de la forme : *Si $C_r$ est vrai, alors $C_s$ devrait être vrai également, dans le cas contraire quelque chose est erroné.*

L'algorithme de vérification d'une contrainte est décrit dans la figure 29. Lorsqu'une contrainte est évaluée, la solution de l'apprenant est examinée tout d'abord par rapport à la condition de pertinence. Ainsi, la condition de satisfaction n'est évaluée que si la solution de l'apprenant satisfait la condition de pertinence (la contrainte peut s'appliquer à la solution proposée). Si cette dernière n'est pas satisfaite alors la contrainte est ignorée. La contrainte est étiquetée comme satisfaite si la solution de l'apprenant vérifie la condition de satisfaction, sinon elle est étiquetée comme violée. Toutes les solutions correctes d'un problème ne violent aucune des contraintes. Une contrainte enfreinte signale une erreur, qui peut provenir d'une connaissance incomplète ou incorrecte.

Les tuteurs intelligents basés sur CBM, en se focalisant sur les contraintes enfreintes, sont capables de générer des rétroactions pédagogiques sans être pour autant capables de résoudre les problèmes. Les rétroactions sont locales aux contraintes car un ou des messages de rétroaction sont attachés à chaque contrainte et l'un d'eux est déclenché lorsque la contrainte est violée. L'ensemble des contraintes satisfaites et violées sont utilisées pour élaborer le modèle de l'apprenant.

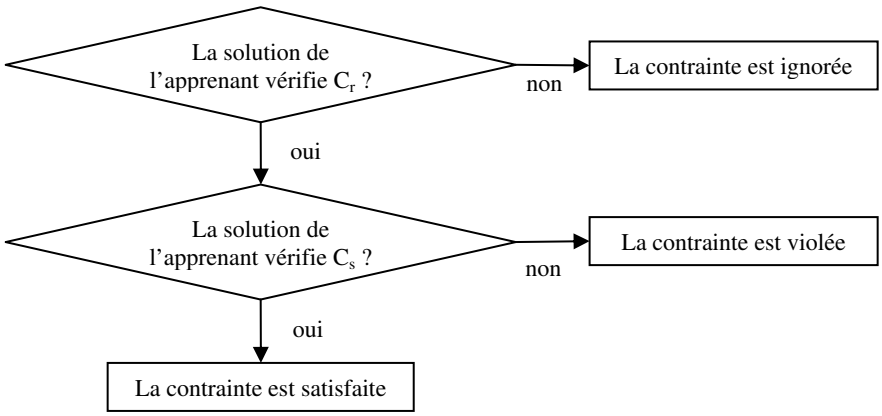



**Figure 29 : Algorithme d'évaluation d'une contrainte [Suraweera 2001]**

Des tuteurs intelligents reposant sur CBM ont été développés dans différents domaines tels que :

- la ponctuation dans CAPIT (***C**apitalisation **A**nd **P**unctuation **I**ntelligent **T**utor*) [Mayo & Mitrovic 2001] ;
- SQL (***S**tructured **Q**uery **L**anguage*) dans SQL-Tutor [Mitrovic 2003] ;
- la normalisation de données dans NORMIT (***NORM**alisation **I**ntelligent **T**utor*) [Mitrovic 2005] ;
- la modélisation Entité-Relation des bases de données dans KERMIT [Suraweera & Mitrovic 2004] ;
- la modélisation orientée objet dans Collect-UML [Baghaei *et al.* 2006] [Baghaei 2007].

Nous nous intéressons au domaine de la modélisation et donc plus particulièrement à KERMIT et à Collect-UML (décrits dans le chapitre 2). Dans ce domaine, peu de contraintes sont exprimables pour analyser le sens de la proposition de l'apprenant. C'est pourquoi ces deux systèmes embarquent, en plus de la base de contraintes, une solution idéale pour chaque problème proposé à l'apprenant. Certaines contraintes expriment ici des écarts acceptables entre la solution idéale et la solution de l'apprenant. L'énoncé d'un problème proposé et la solution idéale sont exprimés directement par l'enseignant comme le fait un apprenant mais dans un environnement dédié générant en sortie un fichier XML (cf. figure 30). Les différents concepts de l'énoncé sont liés par l'enseignant à leurs éléments correspondants dans la solution idéale.

```
(13                ; problem number
10                 ; difficulty
"Draw a UML class diagram for a <E1> School </E1>. A <E1> school </E1> is known by its <E1A1>
name </E1A1>, <E1A2> address </E1A2>, and <E1A3> phone number </E1A3> and <R1> has </R1>
one or more <E2> departments </E2>. Each <E2> department </E2> has a <E2A1> name </E2A1> and
<R2> is assigned </R2> a number of <E3> instructors </E3>. Each <E3> instructor </E3> has a <E3A1>
name </E3A1> and <R3> teaches </R3> several <E4> courses </E4> within the <E2> department </E2>.
Each <E4> course </E4> is known by its <E4A1> name </E4A1> and <E4A2> course ID </E4A2>. A
<E5> student </E5> has a ..."

(("RELATIONSHIPS" "@ R1 aggregation E1 E2 null 1..* null null has @ R2
aggregation E2 E3 null 1..* null null is_assigned @ R3 association E4
E3 1..* 1..* null null teaches ...")
("ATTRIBUTES" "@ E1A1 name E1 String private no @ E1A2 address E1 String
private no @ E1A3 phone_number E1 String private no @ E3A1 name E3
String private no @ E4A1 name E4 String private no @ E4A2 course_ID E4
String private no @ E5A1 name E5 String private no ...")
("METHODS" "@ E1A4 add_student E1 void public no 1 student_ID String null
null null null @ E1A5 emove_student E1 void public no 1 Student_iD
String null null null null ...")
("CLASSES" "@ E1 School concrete @ E3 Instructor concrete @ E4 Course
concrete @ E5 Student concrete @ E2 Department concrete ")
("SUPERCLASSES" "")
("SUBCLASSES" ""))
"13.jpg"
"Schools")
```

**Figure 30 : Un exemple de problème et sa solution idéale dans Collect-UML [Baghaei *et al.* 2006]**

L'analyse des productions de l'apprenant en cours d'activité de modélisation dans ces systèmes se fait par comparaison de la solution idéale (notée IS) avec la solution de l'apprenant (notée SS) grâce à ensemble de contraintes. Les contraintes sont de deux types : d'une part les contraintes syntaxiques décrivant les principes de base du domaine et ne nécessitant pas de comparaison avec la solution idéale et d'autre part les contraintes sémantiques (généralement plus complexes que les contraintes syntaxiques) permettant d'exprimer les écarts tolérés entre la solution idéale et la solution de l'apprenant. KERMIT contient une centaine de contraintes pour le domaine Entité-Relation et Collect-UML contient 88 contraintes syntaxiques et 45 contraintes sémantiques. La figure 31 décrit deux exemples de contraintes utilisées dans l'environnement Collect-UML.

```
(41
 "Check your classes. Each class must have at least one attribute or method."
 ; Relevance condition
 (match SS CLASSES (?* "@" ?class_tag ?*))
 ; Satisfaction condition
 (or-p (match SS ATTRIBUTES (?* "@" ?tag1 ?attr_name ?class_tag ?*))
     (match SS METHODS (?* "@" ?tag2 ?method_name ?class_tag ?*)))
 "classes"
 (?class_tag))

(52
 "Check your inheritance relationships. Some of your subclasses must override one or more methods
defined in the superclass. The ability of a subclass to override a method in its superclass allows a class to
inherit from a superclass whose behavior is similar, and then override methods as needed."
; Relevance condition
 (and (match IS SUPERCLASSES (?* "@" ?c1_tag ?*))
     (match IS SUBCLASSES (?* "@" ?c2_tag ?c1_tag ?*))
     (match IS METHODS (?* "@" ?m1_tag ?name ?c1_tag ?*))
     (match IS METHODS (?* "@" ?m1_tag ?name2 ?c2_tag ?*))
     (match SS SUPERCLASSES (?* "@" ?c1_tag ?*))
     (match SS SUBCLASSES (?* "@" ?c2_tag ?c1_tag ?*))
     (not-p (test SS ("null" ?c1_tag)))
     (not-p (test SS ("null" ?c2_tag)))
     (match SS METHODS (?* "@" ?m1_tag ?name3 ?c1_tag ?*)))
; Satisfaction condition
  (match SS METHODS (?* "@" ?m1_tag ?name4 ?c2_tag ?*))
  "methods"
  (?c1_tag ?c2_tag ?m1_tag))
```

**Figure 31 : Exemples de contraintes exprimées dans Collect-UML [Baghaei *et al.* 2006]**

La contrainte numéro 41 est syntaxique et indique que *toute classe doit comporter au moins un attribut ou une méthode*. Le message de rétroaction pédagogique *« Check your classes. Each class must have at least one attribut or method »* est directement encapsulé dans la contrainte pour être fourni si elle est enfreinte.

La contrainte numéro 52 est sémantique et renvoie au concept de surcharge des méthodes : *Si une classe et sa classe fille possèdent une méthode portant le même nom dans le diagramme solution, et si la classe mère correspondante dans le diagramme de l'apprenant possède la même méthode alors la classe fille doit surcharger cette méthode (avec le même nom).*

Pour faciliter l'analyse des productions et pour ne pas introduire d'ambiguïtés ni de problèmes liés à l'utilisation du langage naturel tels que la synonymie, la polysémie et les fautes d'orthographe ou de frappe, les apprenants sont contraints d'utiliser des expressions de l'énoncé pour construire les différents éléments de leur modèle. De plus, les auteurs insistent sur le fait que les énoncés des problèmes ne doivent pas décrire de concepts du domaine superflus (i.e. non représentés dans la solution idéale) et que les éléments de la solution idéale ne doivent pas être implicites (i.e. ils doivent tous être décrits explicitement dans l'énoncé). Le problème d'appariement des noms des

éléments présents dans les deux modèles (IS et SS) est ainsi fortement simplifié. Cependant, l'activité de modélisation est moins ouverte et plus contrainte pour l'apprenant.

Cette approche se focalise directement sur les erreurs commises par l'apprenant et fournit des rétroactions pédagogiques lorsqu'une contrainte est enfreinte. Le système ne peut donc pas repérer certaines erreurs si elles n'ont pas été définies dans les bases de contraintes. La solution de l'apprenant est considérée comme correcte si aucune contrainte n'est violée. La définition judicieuse des contraintes est très importante pour que les rétroactions soient pertinentes. L'acquisition d'une connaissance demande peu de temps : Mitrovic rapporte qu'il lui faut une moyenne de 1,1 heures de travail pour identifier une contrainte [Mitrovic & Ohlsson 1999]. Néanmoins la définition des contraintes sémantiques peut se révéler ardue si le domaine d'apprentissage et les liens entretenus entre les concepts sont complexes.

### 3.3.3 Discussion

Les deux méthodes d'analyse des réponses de l'apprenant que nous venons de présenter permettent d'évaluer si la réponse de l'apprenant (son diagramme) est correcte ou erronée dans le domaine particulier de la modélisation. Elles ont recours toutes les deux à une solution idéale qu'elles comparent à la solution de l'apprenant. Elles se focalisent sur les erreurs commises par l'apprenant au cours de l'activité de modélisation et orientent les rétroactions directement sur les erreurs identifiées. Un inconvénient majeur des deux approches est leur incapacité à analyser des réponses trop éloignées de la solution idéale. En effet, si l'apprenant définit une solution alternative, qu'elle soit correcte ou non, ces systèmes ne pourront pas fournir de rétroactions pédagogiques sur les parties qu'ils ne peuvent pas analyser. Si la solution alternative est correcte, l'apprenant ne sera pas dérouté par des rétroactions parasites. Toutefois si l'apprenant fournit une solution erronée trop éloignée de la solution idéale, les systèmes seront incapables de produire des rétroactions pertinentes pouvant guider l'apprenant.

Pour pallier ce problème, ces systèmes imposent des contraintes au niveau de l'expression des spécifications textuelles et au niveau de la solution idéale. La tâche de modélisation est ainsi un peu moins ouverte et les appariements des solutions sont plus simples à identifier. La simplification des énoncés par l'élimination des ambiguïtés, des connaissances implicites ou superflues et l'interdiction de représenter des éléments implicites dans les diagrammes exprimant la solution limitent fortement la liberté d'expression de solutions alternatives. L'apprenant est contraint indirectement de construire une solution proche de la solution idéale fournie par l'enseignant.

Ces approches sont néanmoins très bien adaptées dans le contexte de l'apprentissage des bases de la modélisation par des étudiants novices, car celui-ci fait appel à des problèmes relativement simples et permettant de construire des diagrammes de faible taille. L'approche élaborée dans DesignFirst-ITS laisse plus de liberté à l'apprenant au niveau des noms employés dans le diagramme. L'approche CBM, quant à elle, est beaucoup plus générique et souple grâce à l'emploi de contraintes syntaxiques et sémantiques permettant d'analyser la solution de l'apprenant selon différents aspects indépendants (exprimés dans les contraintes).

# CHAPITRE 4

# Les techniques d'appariement de modèles

# PLAN DU CHAPITRE

Nous avons montré dans les chapitres précédents que dans le contexte de l'enseignement de la modélisation, il n'est pas possible de mettre en place un résolveur qui permette de générer de manière automatique une solution pertinente ou un ensemble de solutions valides à partir d'un problème exprimé sous forme de spécifications en langage naturel. De plus, l'activité générale de modélisation et celle plus précise de construction d'un modèle ne suivent pas de méthode définie à l'avance : les actions ou séquences d'actions réalisées par l'apprenant ne permettent pas de déduire immédiatement si l'apprenant élabore des constructions correctes ou erronées par rapport à l'énoncé demandé. Le modèle produit, comme le résultat des actions de l'apprenant, reflète en revanche à un instant donné la représentation que l'apprenant a des spécifications, qu'elle soit correcte, complète ou non. Actuellement, les propositions d'analyse des réponses de l'apprenant se focalisent donc essentiellement sur le modèle construit tout au long de l'activité de modélisation.

Il ressort que l'utilisation d'une solution ou d'un ensemble de solutions valides définies par un expert du domaine est actuellement incontournable pour qu'un système soit capable d'analyser automatiquement les productions des apprenants pendant l'activité de modélisation. Les mauvaises pratiques, les oublis et les erreurs que l'apprenant commet sont identifiés à partir du modèle construit. Les approches CBM et *Curriculum* s'appuient fortement sur le modèle construit par l'apprenant pour produire des aides spécifiques, en le comparant à une référence. À un niveau plus général, ce problème de comparaison de plusieurs modèles créés par des acteurs différents a été étudié en dehors d'un contexte d'apprentissage. Ce processus s'apparente à un processus d'appariement de modèles. L'appariement de modèles a donné lieu à de nombreux travaux spécifiques ou génériques dont les concepts et les techniques peuvent être réutilisés dans notre contexte d'apprentissage, c'est pourquoi nous présentons le problème d'appariement de modèles et les approches classiques qui ont été définies pour le traiter.

Le processus d'appariement joue un rôle central dans les applications des bases de données telles que l'intégration de données, les entrepôts de données (*data warehouses*), le commerce en ligne, la migration et l'évolution de modèles, ou encore le traitement de requêtes sémantiques. L'appariement intervient de manière générale comme une phase essentielle dans la majorité des applications qui manipulent diverses données avec diverses structures dans divers modèles [Zerdazi & Lamolle 2006]. Ces derniers sont le plus souvent représentés par des schémas hétérogènes tels que des schémas relationnels, des schémas orientés objet, des DTD XML... Les données représentant le même concept peuvent donc être structurées de manière totalement différente selon le système de stockage. Le processus d'appariement dans ces contextes est réalisé le plus souvent manuellement et peut être supporté par une interface graphique utilisateur. Il est manifeste que la spécification totalement manuelle des appariements est fastidieuse, chronophage, sujette aux erreurs, et est par conséquent un processus très coûteux. De plus, comme les systèmes deviennent capables de gérer des bases de données et des applications de plus en plus complexes, leurs modèles de données deviennent plus grands et le nombre d'appariements à réaliser augmente davantage. L'effort à réaliser est proportionnel au nombre d'appariements et peut être plus complexe que la linéarité si une évaluation de chaque appariement est requise dans le contexte des autres appariements possibles pour les mêmes éléments. Enfin, de nombreux auteurs s'accordent sur le fait que la tâche d'appariement ne peut être complètement automatique, du fait qu'elle requiert une intervention humaine ayant une expérience dans le domaine de données [Doan *et al.* 2001] [Euzenat & Shvaiko 2007].

Une approche d'intégration plus rapide et moins laborieuse est nécessaire et demande un support automatique ou semi-automatique pour l'appariement. C'est pour cela que de nombreux travaux relatifs à l'appariement ont été définis sur différents types de données telles que les ontologies[11] ou les schémas[12]. Ces modèles le plus souvent hiérarchiques sont assimilables à des graphes.

Nous présentons tout d'abord dans ce chapitre, les définitions et les termes employés dans le cadre du processus d'appariement de modèles puis nous qualifions les différents problèmes généraux d'appariement dans le contexte des graphes. Le choix d'une technique ou d'un ensemble de techniques pour un problème d'appariement n'est pas évident étant donné que de nombreuses techniques d'appariement ont été définies et qu'elles peuvent partager ou non des concepts, des démarches et des critères. Nous présentons plusieurs dimensions et classifications permettant de définir clairement les principaux concepts, caractéristiques et critères employés dans les techniques d'appariement d'ontologies et de schémas. Enfin nous terminons avec quelques approches particulières d'appariement combinant plusieurs techniques.

## 4.1 Généralités sur le problème de l'appariement

Avant de définir les différents problèmes et les techniques d'appariement classiques, il convient de préciser le sens du terme appariement (*matching* ou *match*). Le terme appariement prête à confusion car il peut être aussi bien un processus (ou une tâche) que le résultat de ce même processus. Les définitions de l'appariement varient sensiblement en fonction de ce facteur mais également en fonction de la nature des données à apparier, du domaine, du contexte d'application et de la présentation des résultats. Nous donnons ici quelques définitions que nous avons rencontrées au cours de nos lectures relatives au problème d'appariement et nous caractérisons le problème d'appariement de modèles.

### *4.1.1 Définitions de l'appariement*

L'**appariement de formes ou de motifs** (*pattern matching*) est la mise en correspondance de formes selon un ensemble prédéfini de règles ou de critères [Wikipedia]. L'appariement de formes se ramène à un problème de filtrage. Dans un programme informatique, le **filtrage par motif** (une autre traduction de *pattern matching*) est le fait de vérifier la présence des constituants d'un motif donné. À la différence de la reconnaissance de formes (*pattern recognition*), les motifs sont spécifiés rigidement et concernent conventionnellement des séquences ou des arbres. Le filtrage par motif est utilisé pour vérifier qu'un objet filtré a une structure désirée, pour trouver une structure appropriée, pour retrouver des parties alignées ou pour substituer les motifs reconnus par quelque chose d'autre.

[11] À l'origine, l'Ontologie est une branche de la philosophie dans laquelle les philosophes ont tenté de rendre compte de l'existant de façon formelle. En informatique, une ontologie est comprise comme un système de concepts fondamentaux qui sont représentés sous une forme compréhensible par un ordinateur [Mizoguchi & Bourdeau 2004]. La définition la plus répandue est : « une ontologie est une spécification explicite d'une conceptualisation » [Gruber 1993]. Une ontologie est un système conceptuel qui permet de partager et de réutiliser des concepts grâce à une sémantique computationnelle. Une ontologie fournit typiquement un vocabulaire qui décrit un domaine et une spécification du sens des termes utilisés dans le vocabulaire [Shvaiko & Euzenat 2008]. La notion d'ontologie englobe plusieurs modèles conceptuels et de données : ensembles de termes, classifications, schémas de bases de données...

[12] Un schéma peut être défini comme étant simplement un ensemble d'éléments connectés par une structure quelconque [Rahm & Bernstein 2001]. En pratique, une représentation doit être choisie : les ontologies, les schémas relationnels ou orientés objet de base de données, les schémas XML, les hiérarchies de concepts sont des exemples classiques de schémas étudiés dans les travaux traitant le problème d'appariement de schémas. Les schémas sont le plus souvent des structures de données hiérarchiques.

Dans le cadre des ontologies, l'appariement (*ontology matching*) est le processus de découverte des relations (ou correspondances[13]) entre les entités de différentes ontologies [Euzenat & Shvaiko 2007]. Le terme **alignement** est employé plus communément dans le contexte des ontologies. L'alignement d'ontologies met en évidence les relations sémantiques de plusieurs ontologies à confronter (équivalence, subsomption, incompatibilité, etc.). L'expression des correspondances, appelée aussi alignement, peut par la suite être utilisée par exemple pour fusionner les ontologies, migrer des données entre ontologies ou traduire des requêtes formulées en fonction d'une ontologie vers une autre [Djoufak Kengue *et al.* 2008].

L'**appariement de schémas** (*schema matching*) consiste à trouver les correspondances sémantiques (i.e. appariements) entre deux schémas [Do & Rahm 2007]. L'appariement peut être considéré comme une opération ou un opérateur (*match*) qui prend deux schémas en entrée et produit un *mapping* entre les éléments des deux schémas correspondant sémantiquement les uns aux autres [Rahm & Bernstein 2001].

Les problèmes d'**appariement de graphes** (*graph matching)* consistent à mettre en correspondance les sommets (nœuds) de deux graphes, l'objectif étant généralement de comparer les objets modélisés par les graphes.

Nous avons pu constater que la terminologie relative au problème d'appariement diffère entre les différents domaines et même au sein d'un même domaine. Quelles que soient la nature des modèles à analyser (ontologies, schémas, graphes) et leur représentation retenue en interne, la plupart des modèles en entrée peuvent être transcrits sous forme de graphes. Le but général du processus d'appariement est le même pour tous les types de modèles : **l'identification et la qualification de correspondances entre les éléments[14]**.

### 4.1.2 Résultat du processus d'appariement

Le résultat du processus d'appariement (*match result*), appelé également appariement, alignement ou *mapping* indique quels éléments des modèles en entrée correspondent les uns aux autres. Quelques variations d'interprétation existent entre les notions d'alignement et de *mapping* [Euzenat & Shvaiko 2007]. Un alignement est un ensemble de correspondances entre deux ou plusieurs (cas d'appariements multiples) modèles (par analogie aux alignements de séquence moléculaire). Une **correspondance** est une relation entre plusieurs éléments des modèles. Le terme *mapping* peut être considéré de manière équivalente à celui d'alignement. Néanmoins, dans le contexte des appariements de schémas, le terme alignement est très peu usité et celui de *mapping* est préféré. Cependant, un *mapping* peut aussi être vu comme une version orientée ou dirigée d'un alignement. La définition mathématique requerrait en principe que l'objet d'origine soit égal à son image (i.e. une relation d'équivalence). Un *mapping* est comme une collection de règles (*mapping rules*) toutes orientées dans la même direction, c'est-à-dire d'un modèle vers un autre, et telle que les éléments du modèle source apparaissent une fois au plus. Dans un *mapping*, une *mapping rule* associe un élément d'un

[13] Une correspondance est définie par les deux entités reliées (qui peuvent être des classes, des instances, des propriétés, des termes, mais aussi des combinaisons complexes de ceux-ci), la relation liant ces entités (équivalence, subsomption, incompatibilité, etc.) et si possible une mesure de confiance dans cette correspondance [Euzenat & Shvaiko 2007].

[14] Nous utilisons ici le terme élément pour ne pas présupposer une quelconque représentation des modèles à apparier. Les éléments en fonction du formalisme retenu, peuvent être aussi variés que des objets, des classes, des entités, des instances, des sommets, des relations, des formules...

modèle à un élément de l'autre modèle. Nous conservons la définition du *mapping* comme une version dirigée d'un alignement dans le reste du chapitre.

Le résultat final d'appariement peut associer un ou plusieurs éléments du premier modèle à un ou plusieurs éléments de l'autre modèle et réciproquement. Dans [Rahm & Bernstein 2001] des relations de cardinalité entre les différents éléments appariés sont introduites pour qualifier les appariements produits. Elles correspondent à quatre cas de figure : 1:1 (un élément associé à un élément : couplage), 1:n (un élément associé à n éléments), n:1 (n éléments associés à un élément), ou m:n (m éléments associés à n éléments). Les trois derniers cas de figures correspondent à des appariements multiples (multivoques) d'éléments. De plus, un élément d'un modèle peut intervenir dans zéro, une ou plusieurs correspondances d'éléments de l'alignement produit entre deux modèles : la cardinalité est qualifiée de globale à cette échelle. En outre, à l'intérieur d'une correspondance individuelle, un ou plusieurs éléments d'un diagramme peuvent être appariés à un ou plusieurs éléments de l'autre modèle : la cardinalité de l'appariement est qualifiée ici de locale. Les cas de cardinalité globale qui concernent tous les éléments appariés sont orthogonaux aux cas des éléments appariés individuellement. C'est-à-dire qu'un résultat d'appariement peut avoir une cardinalité globale 1:n en combinaison avec des cardinalités locales d'appariement 1:1 ou 1:n.

La plupart des approches d'appariement sont restreintes à la cardinalité locale 1:1 car elles sélectionnent comme candidat d'appariement pour un élément d'un modèle, l'élément le plus similaire dans l'autre modèle [Do *et al.* 2002]. Les alignements produits ont donc en général une cardinalité globale 1:1 ou 1:n. Plus d'efforts et l'utilisation de critères plus sophistiqués sont nécessaires pour générer des appariements locaux et globaux de cardinalité n:1 et m:n. L'obtention d'alignements d'éléments de cardinalité globale m:n requiert par exemple de prendre en compte les structures agençant les éléments dans les modèles.

### 4.1.3 Processus d'appariement

Le processus d'appariement peut être défini sous forme d'une « boite noire ». Cette boite noire est une opération qui détermine l'alignement *A'* pour une paire de modèles *m* and *m'*. Quelques autres paramètres peuvent étendre la définition du processus : l'utilisation en entrée d'un alignement *A* qui est complété par le processus, des paramètres d'appariement (par exemple des poids et des seuils), des ressources externes utilisées par le processus d'appariement. Techniquement, le processus d'appariement peut être vu comme une fonction *f* qui, à partir d'une paire de modèles à apparier *m* et *m'*, un alignement en entrée *A*, un ensemble de paramètres *p* et un ensemble de ressources externes *r* (par exemple des thesaurus), retourne un alignement *A'* entre ces modèles :

$$A' = f(m, m', A, p, r)$$

Le processus d'appariement peut être représenté schématiquement comme c'est le cas dans la figure suivante :

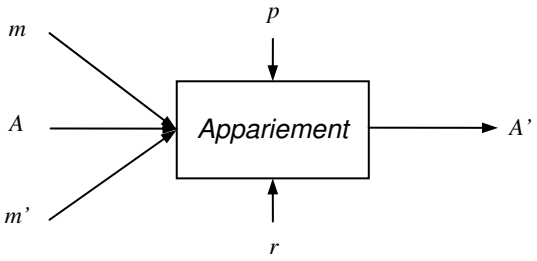



**Figure 32 : Processus d'appariement**

Il peut être utile dans certains cas de considérer spécifiquement l'appariement de plus de deux modèles avec le même processus [Euzenat & Shvaiko 2007]. Ce processus est nommé dans ce cas appariement multiple (*multiple matching*). Le processus d'appariement multiple peut alors être vu comme une fonction *f* qui à partir d'un ensemble de modèles à apparier $\{m_1, ...m_n\}$, un alignement en entrée *A*, un ensemble de paramètres *p* et un ensemble de ressources, retourne un alignement *A'* entre ces modèles.

$$A' = f(m_1, ...m_n, A, p, r)$$

### 4.1.4 Problèmes d'appariement et mesure de similarité

Nous venons de présenter le processus d'appariement sous forme d'une boite noire. En interne, l'appariement de plusieurs modèles et par conséquent de leurs constituants implique qu'ils soient comparés selon divers critères. Le problème de comparaison de plusieurs objets repose notamment sur l'évaluation de leurs similarités ou de leurs différences. La **mesure de la similarité** de deux objets consiste à identifier et à quantifier leurs points communs alors que la **mesure de distance** entre deux objets est l'identification et la quantification de leurs différences. Dans le contexte des graphes et donc des modèles assimilables à des graphes (c'est le cas pour les majorités des modèles manipulables dans les approches d'appariement de schémas ou d'ontologies), distance et similarité sont deux concepts duaux qui renvoient à un objectif commun. Le calcul de la distance ou de la similarité de deux structures de graphes permet habituellement de trouver le **« meilleur » appariement** des sommets d'un graphe, c'est-à-dire celui qui préserve le plus de caractéristiques des sommets et des arcs [Sorlin *et al.* 2007]. Vu sous un autre angle, l'appariement est un problème d'optimisation où l'objectif est de trouver un alignement induisant la distance la plus faible entre les objets comparés.

Un appariement de graphes est dit **univoque** (*univalent*) lorsque chaque sommet d'un graphe est associé avec au plus un sommet de l'autre graphe (cardinalité d'appariement 1:1). Un appariement est dit **multivoque** lorsque chaque sommet d'un graphe peut être associé à un ensemble de sommets de l'autre graphe [Sorlin *et al.* 2007]. Un appariement multivoque d'un sommet d'un graphe avec plusieurs autres sommets de l'autre graphe se nomme généralement **éclatement** de sommets (cardinalité d'appariement 1:n). Inversement, un appariement multivoque de plusieurs sommets associés à un seul sommet de l'autre graphe est une **fusion** de sommets (cardinalité d'appariement n:1).

Différents problèmes courants d'appariement de graphes existent. Les mieux connus sont les appariements de graphes **univoques exacts** lorsque toutes les caractéristiques des sommets et des arcs sont préservées dans l'appariement ; c'est le cas notamment dans l**'isomorphisme de graphes et de sous-graphes** qui permettent respectivement de vérifier si deux graphes sont structurellement identiques (relation d'équivalence) ou si un graphe est « inclus » dans un autre (relation d'inclusion). Les appariements peuvent être **univoques tolérants aux erreurs** lorsque toutes les caractéristiques des sommets et des arcs ne peuvent pas être préservées lors de l'appariement (les objets appariés ne sont pas équivalent en tout point) : la recherche du **plus grand sous-graphe commun** (la plus grande partie commune à deux graphes) ou le calcul de **la distance d'édition de graphes** (le plus petit nombre d'opérations à effectuer pour transformer un graphe en un autre) en sont deux exemples.

Dans les cas plus complexes où une mise en correspondance univoque (un sommet avec zéro ou un sommet) ne suffit pas, il est nécessaire de mettre en place une mesure de similarité **multivoque tolérante aux erreurs**. Cette mesure est nécessaire dans les problèmes concernant la comparaison d'objets décrits à différents niveaux de granularité et où la recherche d'appariements multivoques (éclatements et fusions) est obligatoire. C'est le cas notamment dans des problèmes d'appariement d'une image bruitée à un modèle schématique (imagerie du cerveau humain par exemple) ou dans des problèmes d'appariements d'objets sur/sous segmentés. Les problèmes de recherche de composants complexes ou *patterns* dans des modèles en génie logiciel nécessitent également ce type de mesure pour restructurer ensuite les modèles de manière automatique.

## 4.2 Classifications des approches d'appariement

Les choix d'implantation d'une approche d'appariement dépendent des caractéristiques des modèles, de l'environnement d'appariement et de l'utilisation attendue des résultats. Depuis ces dix dernières années, de nombreuses techniques d'appariement ont été définies et appliquées sur les ontologies, les schémas et les graphes dans le cadre de diverses applications. Ces travaux reposent sur des concepts, des algorithmes, des critères et des ressources externes qui peuvent varier fortement et n'ont pas le même degré de généricité et de réutilisabilité.

Une étude détaillée des approches existantes est nécessaire pour pouvoir choisir quelle technique ou ensemble de techniques peut s'appliquer le mieux à un problème et à un contexte donné. Plusieurs classifications et taxonomies couvrant un grand nombre d'approches d'appariement existantes (d'ontologies et de schémas) ont été définies et présentées dans [Rahm & Bernstein 2001], [Shvaiko 2004] et [Euzenat & Shvaiko 2007]. Il ressort de ces travaux que les approches d'appariement peuvent être analysées et comparées selon de nombreuses dimensions indépendantes. Ces dimensions sont très utiles pour les programmeurs et les chercheurs qui souhaitent développer de nouveaux algorithmes, réutiliser une approche existante ou un ensemble de techniques d'appariement le plus adapté pour traiter un problème d'appariement avec des caractéristiques précises.

### 4.2.1 Classification préliminaire des approches d'appariement

Une première classification primaire des approches peut être réalisée en partant de la définition de l'appariement sous la forme d'une fonction *f* (cf. partie 4.1.3). Les dimensions sont relatives dans ce cas aux entrées, aux caractéristiques liées au processus d'appariement et aux sorties.

Les dimensions relatives aux **entrées** concernent tout d'abord les types et la représentation en interne des modèles sur lesquels les algorithmes opèrent. En effet, les modèles conceptuels ou de données utilisés en entrée peuvent être variés (par exemple des modèles relationnels, orientés objet, entité-relation...). Ensuite, les informations utilisées lors de l'appariement ne sont pas les mêmes dans tous les algorithmes. De plus, à partir des mêmes modèles en entrée, les techniques d'appariement n'exploitent pas toujours toutes les constructions disponibles. En général, certains algorithmes se focalisent sur les étiquettes assignées aux éléments, d'autres considèrent leur structure interne, les types des attributs, et enfin d'autres encore prennent en compte leurs relations avec d'autres éléments.

Beaucoup de modèles à apparier sont structurés de manière hiérarchique (ils reposent par exemple sur des relations de contenance). En réalisant un appariement sur des structures hiérarchiques, un algorithme peut parcourir la structure de haut en bas (*top-down*) ou de bas en haut (*bottom-up*). Un algorithme *top-down* est en général moins coûteux qu'un algorithme *bottom-up*, car l'appariement à un niveau élevé de la structure du modèle restreint les choix d'appariement des structures plus fines seulement sur des combinaisons avec les appariements parents [Madhavan *et al.* 2001]. Cependant, un algorithme descendant peut être erroné si les structures de haut niveau sont très différentes, et cela même si les éléments avec une granularité plus faible s'apparient bien. Au contraire, un algorithme *bottom-up* compare toutes les combinaisons possibles des éléments de granularité fine et cherche ensuite les appariements à ce niveau même si les structures intermédiaires et de haut niveau diffèrent de manière considérable.

Le processus d'appariement peut être analysé sur ses propriétés générales et en particulier selon la nature exacte ou approximative de son calcul. Les **algorithmes exacts** calculent la solution exacte d'un problème (i.e. ils garantissent la découverte de tous les alignements possibles) alors que les **algorithmes approchés** tendent à être incomplets et sacrifient l'exactitude pour la performance [Ehrig & Sure 2004]. Une autre dimension pour analyser les processus d'appariement est basée sur la manière dont ils analysent les données en entrée (des données intrinsèques aux modèles, des ressources externes ou des théories sémantiques...).

En dehors des informations que les systèmes d'appariement exploitent et de la manière dont ils les manipulent, l'autre classe importante de dimensions concerne la forme du résultat que ces systèmes produisent. La forme d'un alignement peut avoir une importance notamment pour savoir si c'est un appariement univoque ou multivoque, si c'est une correspondance finale ou non. Le résultat sera différent si un système délivre une réponse graduée (une correspondance avec un degré de confiance de 90% ou une probabilité de 4/5 par exemple) ou une réponse tout ou rien (identification d'une correspondance ou non). Une autre dimension concerne les relations entre les éléments mis en correspondance : la majorité des systèmes se focalisent sur l'équivalence alors que quelques systèmes sont capables de qualifier le résultat de manière plus expressive (par exemple des relations d'équivalence, de subsomption et d'incompatibilité).

D'un point de vue plus centré sur l'utilisateur final, trois catégories ont être proposées dans [Shvaiko & Euzenat 2005] pour préciser la manière dont le résultat du processus d'appariement est considéré. Les alignements peuvent alors être vus comme :

- **des solutions** : cette catégorie couvre les techniques algorithmiques qui considèrent qu'un alignement est une solution au problème d'appariement. Le problème d'appariement est comme un problème d'optimisation et l'alignement produit est une solution à ce problème ;
- **des théorèmes** : les systèmes de cette catégorie comptent sur la sémantique et exigent à l'alignement de la satisfaire. Le problème d'appariement est exprimé sous forme de termes sémantiques ;
- **des indications de ressemblance** : cette catégorie se réfère aux algorithmes qui produisent seulement des indications utilisables par un utilisateur pour sélectionner les alignements finaux.

### 4.2.2 Classification de Euzenat et Shvaiko

Pour classer les techniques élémentaires d'appariement, Shvaiko et Euzenat dans [Shvaiko & Euzenat 2005] pour l'appariement de schémas puis dans [Euzenat & Shvaiko 2007] pour l'appariement d'ontologies ont introduit deux classifications synthétiques reposant sur la plupart des propriétés des dimensions d'appariement. Ces deux classifications sont représentées sous forme de deux arbres qui partagent leurs feuilles (cf. figure 33).

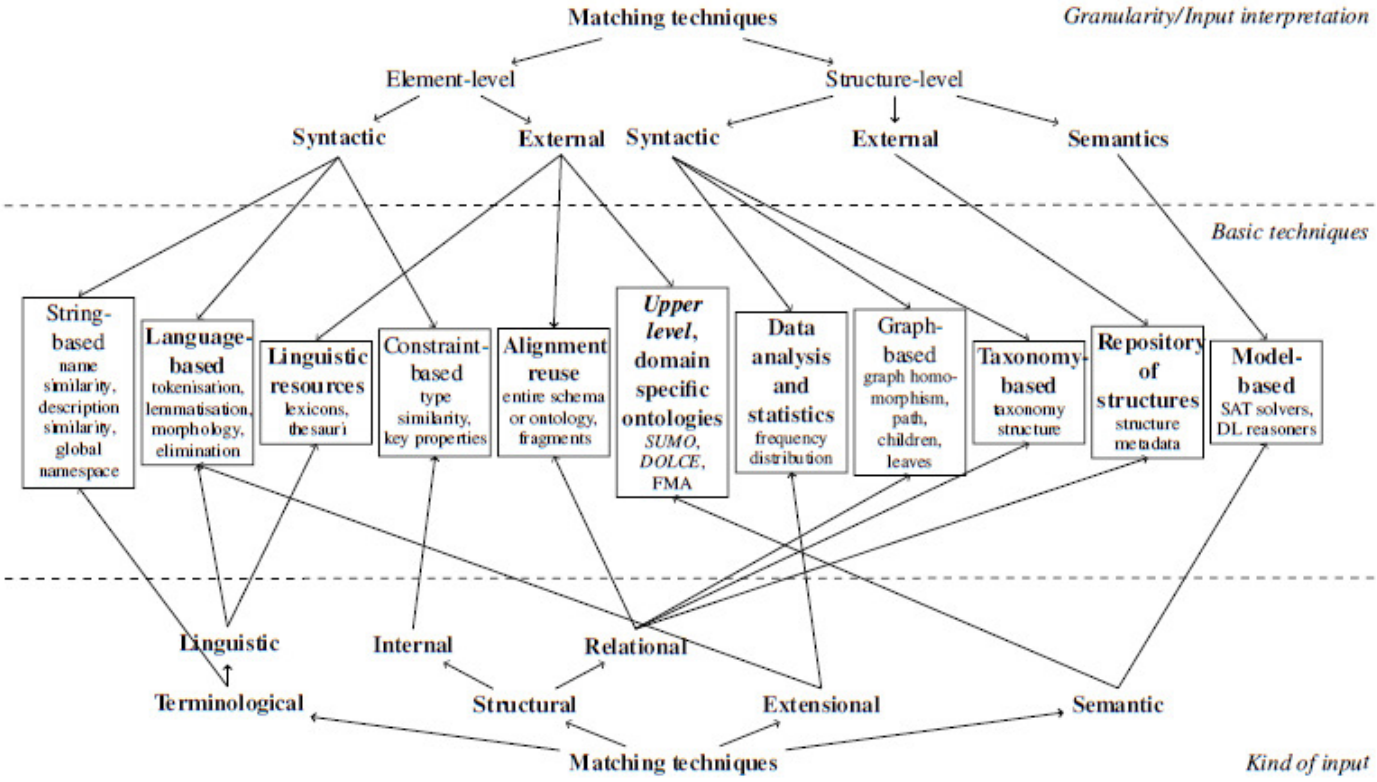



**Figure 33 : Classification des approches d'appariement élémentaires [Euzenat & Shvaiko 2007]**

Les feuilles de la classification de Euzenat et Shvaiko représentent les classes de techniques élémentaires d'appariement et leurs exemples concrets. Les deux classifications sont les suivantes :

- *Granularity/Input Interpretation classification :* cette classification est basée sur la granularité des appariеurs (niveau élément ou structure) et ensuite sur la manière dont les techniques interprètent les informations fournies ;
- *Kind of Input classification :* cette deuxième classification est basée sur la catégorie d'entrées qui est utilisée par des techniques élémentaires d'appariement.

La classification de la figure 33 peut être lue aussi bien de manière descendante (focalisation sur comment les techniques interprètent l'entrée) que de manière ascendante (focalisation sur les catégories d'objets manipulés). Les distinctions entre les classes de techniques élémentaires de la couche centrale du schéma (*Basic Techniques*) sont motivées par la manière dont une technique interprète l'entrée dans chaque cas concret. Par exemple, une étiquette peut être interprétée aussi bien comme une chaîne de caractères, un mot ou une phrase en un langage naturel, une hiérarchie peut être considérée comme un graphe (un ensemble de nœuds reliés par des arcs) ou une taxonomie (un ensemble de concepts organisé par une relation préservant l'inclusion).

La couche *Kind of input* du schéma est relative au type de l'entrée considérée par une technique particulière :

- Le premier niveau est séparé en fonction des familles de données que les algorithmes manipulent : des chaînes de caractères (*terminological*), des structures (*structural*), des modèles (*semantics*) ou encore des instances de données (*extensional*). Les deux premières familles proviennent directement de la description des modèles à apparier. Le troisième requiert une interprétation plus poussée des modèles du point de vue de la sémantique et repose généralement sur des raisonneurs pour déduire les correspondances. La dernière famille renvoie à la population d'une ontologie.
- Le second niveau subdivise les familles si nécessaire : les méthodes terminologiques peuvent être par exemple basées sur l'analyse des chaines de caractères (*linguistic*). La catégorie des méthodes structurelles est éclatée quant à elle en deux types particuliers de méthodes : d'une part celles qui considèrent la structure interne (*internal*) des éléments (leurs attributs et leurs types) et celles qui prennent en compte les relations avec les autres éléments (*relational*).

Nous allons maintenant exposer succinctement les particularités des techniques individuelles en fonction des critères permettant de les classer du point de vue de la granularité et de l'interprétation des entrées. Nous nous focalisons plus particulièrement sur les différences entre les approches de niveau élément et structure ainsi que sur leurs subdivisions (syntaxique, externe ou sémantique).

#### 4.2.2.1 Techniques de niveau élément ou structure

Les techniques de niveau élément considèrent les éléments des modèles ou leurs instances sans prendre en compte leurs relations avec les autres éléments ou leurs instances. Les techniques de niveau structure considèrent quant à elle les éléments ou leurs instances en comparant leurs relations avec les autres éléments ou instances.

Les **approches linguistiques** sont des approches de niveau élément qui utilisent les noms et quelquefois leurs descriptions textuelles (i.e. des mots ou des phrases commentant les éléments) pour identifier les éléments équivalents sémantiquement. Les appareurs basés sur les noms mettent en correspondance les éléments des modèles ayant des noms égaux ou similaires. La définition et la mesure de la similarité des noms peut être réalisée par différents moyens incluant l'égalité stricte ou canonique des représentations des noms après un prétraitement ou non (par exemple le traitement de préfixes et de suffixes particuliers), l'égalité des synonymes et des hyperonymes (en utilisant des ressources auxiliaires telles que des thesaurus et des dictionnaires), la similarité de noms basée sur les sous chaînes communes, la distance d'édition, la prononciation et enfin la prise en compte directe d'appariements fournis par l'utilisateur. Les modèles contiennent souvent des commentaires en langage naturel pour exprimer la sémantique attendue des éléments (par exemple des commentaires ou des notes). Ces commentaires peuvent aussi être évalués pour déterminer la similarité entre les éléments en y extrayant des mots clés ou en recourant à des techniques plus sophistiquées de compréhension du langage naturel.

Les modèles contiennent souvent des contraintes pour définir les types de données et de valeurs, l'unicité, l'optionalité, les types de relations, les cardinalités, etc. Si chaque modèle en entrée contient de telles informations alors elles peuvent être utilisées par un appareur pour déterminer la similarité des éléments des schémas [Larson *et al.* 1989]. Les **techniques basées sur des contraintes** se focalisent sur les contraintes internes des modèles. Par exemple, la similarité peut être appuyée sur l'équivalence des types de données et de domaines, sur des caractéristiques (par exemple unique, clé primaire), sur la cardinalité des relations (par exemple une relation binaire), ou encore sur la relation de classification.

En dehors des ressources externes linguistiques (le plus souvent des dictionnaires ou des thesaurus), certaines techniques de niveau élément enregistrent les alignements des modèles calculés précédemment et les réutilisent pour améliorer l'appariement d'autres modèles. La réutilisation des résultats d'appariement est motivée par le fait que beaucoup de modèles à apparier sont similaires à des modèles déjà évalués, notamment s'ils appartiennent au même domaine d'application. La réutilisation peut être mise en place sous forme d'appareurs se focalisant sur les modèles dans leur intégralité ou sur des fragments de modèles. La réutilisation à l'échelle de fragments est plus efficace en général.

À l'échelle des structures, les techniques basées sur les graphes et les taxonomies reposent sur des algorithmes considérant les modèles en entrée comme des graphes. Dans les premières, les modèles sont transcrits sous forme de graphes étiquetés et orientés. Généralement, la comparaison des similarités entre des paires de nœuds est focalisée sur l'analyse de leur position à l'intérieur des graphes à comparer. Le concept sous-jacent est que si deux nœuds sont similaires, leurs voisins le sont en quelque sorte aussi. Le calcul de la similarité dans certain algorithme peut être orienté plus spécifiquement sur les nœuds enfants, les feuilles ou les relations entretenues entre les nœuds. Les techniques basées sur les taxonomies considèrent en général seulement la relation de spécialisation entre les concepts. Les liens *is-a* (autrement appelés liens d'héritage ou de classification) connectent des termes qui sont déjà similaires. Par conséquent leurs voisins peuvent être également similaires d'une manière ou d'une autre. Ces différentes techniques prennent en compte le voisinage et le contexte des éléments lors du processus d'appariement.

D'autres techniques de niveau structure utilisent des entrepôts de structures (*repository of structures*) sauvegardant des schémas ou des ontologies et leurs fragments ensemble avec leurs similarités entre eux (par exemple, des coefficients compris dans un intervalle [0, 1]). Ces entrepôts ne mémorisent que les similarités, pas les alignements (contrairement aux techniques de réutilisation des alignements). Le terme structures renvoie aux schémas, aux ontologies ou à leurs fragments. Lorsque de nouvelles structures doivent être appariées, leurs similarités sont vérifiées en premier par rapport à celles contenus dans l'entrepôt. Le but est d'identifier les structures qui sont suffisamment similaires pour nécessiter un appariement plus précis, ou réutiliser des alignements préexistants. La détermination de similarités entre des structures demande beaucoup moins de calculs que leur appariement détaillé.

Enfin, certaines techniques de niveau structure peuvent recourir à des algorithmes qui analysent l'entrée en se focalisant sur l'interprétation de sa sémantique. Pour cela, ils utilisent en général des méthodes déductives comme par exemple des techniques de raisonnement de satisfaisabilité propositionnelle (SAT) et de logique descriptive (*Description Logic*).

#### 4.2.2.2 Techniques syntaxiques, externes ou sémantiques

La caractéristique clé des techniques d'appariement syntaxique est qu'elles interprètent l'entrée en fonction de ses constituants seulement en utilisant des algorithmes clairement définis. Les techniques externes exploitent des ressources auxiliaires (externes aux modèles à apparier) d'un domaine ou des connaissances communes pour interpréter les modèles en entrée. Ces ressources peuvent être aussi bien des saisies d'un acteur humain que des thesaurus exprimant des relations entre les termes. Le point clé des techniques sémantiques est qu'elles utilisent la sémantique formelle pour interpréter l'entrée et justifier leurs résultats (par exemple un modèle théorique de la sémantique).

En dehors des informations dont disposent les appariers, l'autre dimension importante est la manière dont ils les exploitent et produisent le résultat [Shvaiko 2004]. En se basant sur ce critère, les systèmes d'appariements peuvent être **syntaxiques** (*syntactic*) ou **sémantiques** (*semantic*). Les approches d'appariement syntaxiques n'analysent pas directement la signification des termes, et par conséquent la sémantique. Dans ces approches, les correspondances sémantiques sont déterminées en utilisant des mesures de similarité syntaxiques, habituellement exprimées dans un intervalle [0..1] (par exemple à l'aide de coefficients de similarité ou de mesures de confiance) ou des techniques dirigées par la syntaxe qui considèrent les étiquettes comme des chaînes de caractères par exemple. La première distinction clé entre une approche syntaxique et une approche sémantique est que dans cette dernière les alignements entre les éléments sont calculés en évaluant des relations sémantiques. Ces relations entre les concepts des nœuds[15] de deux modèles peuvent être par exemple des relations d'équivalence (=), « plus général » ($\subseteq$), « moins général » ($\supseteq$), ou de disjonction ($\perp$). Si aucune de ces relations ne s'applique alors une relation spéciale notée *idk* (*I don't know* ou (?)) est retournée en général. La seconde différence est que les relations sémantiques sont déterminées en analysant le sens qui est véhiculé dans les éléments et la structure des modèles (les concepts des nœuds au lieu de ceux des étiquettes utilisés dans les approches syntaxiques). Ces idées sont représentées schématiquement dans la figure 34.

[15] [Shvaiko 2004] [Giunchiglia *et al.* 2005] introduisent les notions de « concept d'une étiquette » (*concept of/at a label*) et de « concept d'un nœud » (*concept of/at a node*) pour l'appariement sémantique d'arbres. Le concept d'une étiquette est vu comme l'ensemble des documents relatifs à ce que signifie l'étiquette dans le monde. Le concept d'un nœud est l'ensemble des documents que quelqu'un voudrait classer sous ce nœud étant donné qu'il a certaines étiquettes et qu'il est à une certaine position dans l'arbre. L'analyse des concepts des nœuds porte sur le sens des positions que les étiquettes ont dans l'arbre. Cette analyse permet de capturer la connaissance résidant dans la structure des modèles.

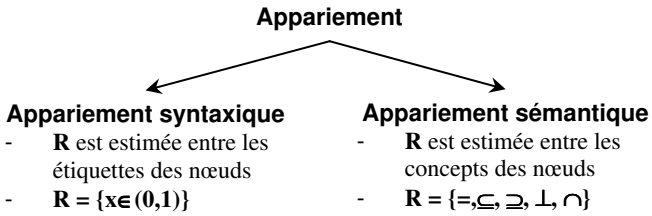



**Figure 34 : Appariement : Syntaxe vs. Sémantique [Shvaiko 2004]**

Il est intéressant de remarquer que la majorité des approches d'appariement sont des variations de l'appariement syntaxique et qu'elles utilisent le plus souvent des informations auxiliaires lors du processus d'appariement.

### *4.2.3 Utilisation de différents appariеurs*

Rahm et Bernstein dans [Rahm & Bernstein 2001] avancent qu'une implantation d'un processus d'appariement peut utiliser des algorithmes multiples d'appariement ou appariеurs. Il est peu probable qu'un système n'utilisant qu'une seule approche individuelle relève autant de candidats à l'appariement corrects qu'un système reposant sur une combinaison de plusieurs approches. De plus, la combinaison de plusieurs approches ouvre la possibilité de les évaluer simultanément ou dans un ordre spécifique. Pour cela, il faut sélectionner les appariеurs en fonction du domaine d'application et des types de modèles. Mais l'utilisation de plusieurs appariеurs induit deux sous problèmes. Premièrement, il y a la réalisation des appariеurs individuels qui calculent un appariement en se focalisant sur un seul critère d'appariement. Deuxièmement, il faut définir la manière dont les appariеurs individuels sont combinés dans le système complet. La combinaison peut se faire sous la forme d'un **appariеur hybride** qui intègre des critères multiples d'appariement (par exemple une égalité des noms et une égalité des types) ou bien sous la forme d'un **appariеur composite** qui combine les résultats produits par différents algorithmes individuels d'appariement. Cette classification sous forme d'appariеurs individuels et combinés est schématisée dans la figure 35.

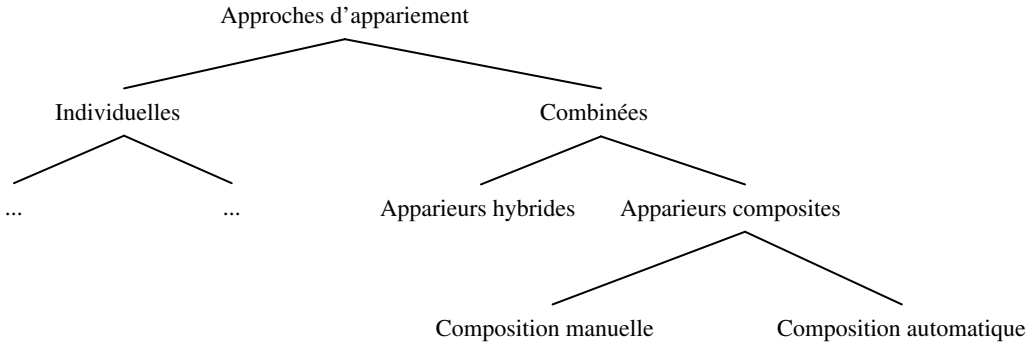



**Figure 35 : Approches individuelles et combinées d'appariement [Rahm & Bernstein 2001]**

Les appariеurs hybrides combinent directement plusieurs approches d'appariement pour déterminer les candidats à l'appariement en se focalisant sur des critères et des sources d'informations multiples. Ils peuvent fournir des appariements meilleurs et avec une meilleure performance qu'en exécutant de manière séparée plusieurs appariеurs (réduction du nombre de passes effectuées sur les modèles). L'efficacité peut être améliorée également car les candidats faibles de l'appariement (ceux qui mettent en correspondance seulement un ou quelques critères) peuvent être filtrés plus tôt et car les appariements complexes requièrent la considération jointe de critères multiples pour être résolus.

Les appariеurs composites combinent les résultats évalués indépendamment par plusieurs appariеurs dont des appariеurs hybrides. Cette capacité à combiner les appariеurs est plus flexible que la combinaison « dure » de techniques particulières d'appariement exécutées simultanément ou dans un ordre fixe dans les appariеurs hybrides. Par opposition, un appariеur composite sélectionne à partir d'un ensemble d'appariеurs modulaires ceux basés, par exemple, sur le domaine d'application ou le formalisme des modèles considérés. De plus, un appariеur composite peut permettre un ordonnancement flexible des appariеurs en les exécutant aussi bien simultanément que séquentiellement. Dans ce dernier cas, le résultat d'un premier appariеur est consommé et étendu par un second appariеur pour accomplir une amélioration itérative du résultat d'appariement.

La sélection des appariеurs, la détermination de leur ordre d'exécution et de la combinaison des résultats d'appariement déterminés indépendamment peuvent être faites aussi bien automatiquement lors de l'implantation du système, que manuellement par un acteur humain. Une approche automatique peut réduire le nombre d'interactions avec l'utilisateur, mais elle est difficile à mettre en place sous forme d'une solution générique adaptable à différents domaines d'application. De manière alternative, un utilisateur peut directement sélectionner les appariеurs à exécuter, leur ordonnancement et comment combiner les résultats. Une telle approche est plus simple à implanter et laisse plus de contrôle à l'utilisateur.

Beaucoup d'auteurs s'accordent sur le fait qu'une approche entièrement automatique est impossible pour gérer le problème d'intégration de schémas [Sheth & Larson 1990]. Dans tous les cas, l'interaction avec l'utilisateur est toujours nécessaire car l'implantation d'un système d'appariement détermine des candidats à l'appariement que l'utilisateur peut accepter, rejeter ou changer. De plus, il faut noter que la majorité des approches d'appariement sont implantées sous forme d'un appariеur hybride dédié à un problème particulier d'appariement.

## 4.3 Quelques exemples d'approches d'appariement

Nous venons de présenter différentes dimensions de classification tirées de la littérature permettant de définir et de comparer les approches d'appariement de modèles les unes par rapport aux autres. Nous allons maintenant nous concentrer sur quelques approches récentes d'appariement de schémas et de graphes au regard de ces dimensions de classification. Pour couvrir la majeure partie des dimensions de classification, nous avons retenues cinq approches distinctes. Le cas échéant, nous précisons brièvement comment ces approches sont implantées dans des systèmes d'appariement.

### 4.3.1 Cupid

Cupid [Madhavan *et al.* 2001] est un appariеur hybride de niveau élément et structure. Il utilise des techniques linguistiques et structurelles qui permettent de calculer des coefficients de similarité. Cet appariеur a recours également à des thesaurus spécifiques au domaine des modèles à apparier. Les modèles en entrée sont encodés en interne comme des graphes. Les nœuds sont parcourus à la fois de bas en haut et de haut en bas. L'algorithme d'appariement est constitué de trois phases et est exécutable uniquement sur des structures arborescentes. La première phase (un appariement linguistique) évalue des coefficients de similarité entre les noms des éléments selon de nombreux critères. La seconde phase (un appariement structurel) évalue des coefficients de similarité structurels relatifs à la similarité des contextes dans lesquels les éléments unitaires des modèles interviennent. La similarité de deux éléments à la racine des structures est basée sur leur similarité linguistique et sur la similarité de l'ensemble de leurs feuilles. La focalisation sur l'ensemble des feuilles repose sur l'hypothèse que beaucoup d'informations sont représentées dans les feuilles et que les feuilles ont moins de variation dans les modèles que les structures internes. La deuxième phase est conclue par le calcul d'une moyenne pondérée des similarités linguistiques et structurelles pour les paires d'éléments comparées. La troisième phase est la génération effective de l'alignement des éléments. Elle utilise les moyennes pondérées de similarité pour proposer un alignement. Elle génère notamment l'alignement final en choisissant les paires d'éléments des modèles ayant les coefficients de similarité pondérés plus grand qu'un seuil. Cupid retourne ainsi des correspondances de niveau élément ayant une cardinalité locale 1:1 et globale 1:n.

### 4.3.2 Similarity flooding

Melnik *et al.* présentent un algorithme d'appariement de graphe nommé *Similarity Flooding* (SF) et expliquent son utilisation pour l'appariement de schémas dans [Melnik *et al.* 2002]. L'approche convertit un schéma en un graphe orienté étiqueté et utilise un algorithme itératif convergeant vers un « point fixe » (i.e. lorsque les similarités de tous les éléments des modèles sont stabilisées ou lorsque qu'un certain nombre d'itérations est atteint) pour déterminer les correspondances des nœuds des graphes à apparier. Pour évaluer les similarités, cette proposition repose sur l'hypothèse que les éléments de deux modèles distincts sont similaires quand leurs éléments adjacents sont similaires. En d'autres termes, une partie de la similarité de deux éléments est propagée à leurs voisins respectifs. L'algorithme SF est implanté sous la forme d'un composant d'un outil générique de manipulation des schémas. En plus de l'algorithme SF, cet outil supporte d'autres dispositifs tels qu'un appariеur de noms, des convertisseurs de schémas, et des filtres qui sont combinés avec des scripts. Un script d'appariement typique commence par convertir les deux schémas fournis en entrée dans la représentation de graphe interne. Ensuite l'appariеur de noms est utilisé pour suggérer un *mapping* initial au niveau des éléments qui est nourri pour l'appariеur structurel SF. Contrairement à de nombreuses approches d'appariement de schémas, SF n'exploite pas de relations terminologiques dans un dictionnaire externe, mais il relève entièrement de la similarité des chaînes entre les noms des éléments. Dans la dernière étape, plusieurs filtres peuvent être spécifiés et appliqués pour sélectionner les sous-ensembles de résultats d'appariement produits par l'appariеur structurel. Cette approche produit un appariement local univoque et multivoque au niveau global entre les éléments des schémas.

### 4.3.3 S-Match

S-Match [Giunchiglia *et al.* 2005] [Giunchiglia *et al.* 2007] est un système d'appariement hybride de niveaux élément et structure qui prend en entrée deux structures assimilables à des graphes (par exemple des schémas ou des ontologies) et retourne en sortie les relations sémantiques (par exemple d'équivalence ou de subsomption) entre les nœuds des graphes qui correspondent sémantiquement les uns aux autres. Les relations sont déterminées en analysant le sens (les concepts au lieu des étiquettes) qui est encodé dans les éléments et les structures de schémas ou d'ontologies. En particulier, les étiquettes des nœuds, écrites en langage naturel, sont traduites sous forme de formules propositionnelles encodant explicitement leur sens. Cette transformation est effectuée pour transformer un problème d'appariement en un problème propositionnel de satisfaisabilité (un problème SAT[16]), qui peut être résolu de manière complète en utilisant l'état de l'art des algorithmes et des systèmes de satisfaction. S-Match a été conçu et développé sous forme d'une plateforme pour l'appariement sémantique, c'est-à-dire un système fortement modulaire avec un noyau de calcul des relations sémantiques où des composants individuels peuvent être ajoutés, retirés ou modifiés. La librairie de S-Match contient treize appariers de niveau élément et trois de niveau structure.

### 4.3.4 COMA

COMA (***C****ombinaison of* ***MA****tchnig* ***A****lgorithms*) [Do & Rahm 2002] suit une approche composite qui fournit une bibliothèque extensible de différents appariers et propose plusieurs manières pour combiner les résultats d'appariement. Les appariers exploitent des informations des schémas, telles que des propriétés structurelles et relatives aux éléments. De plus, un appariur spécial est fourni pour réutiliser les résultats provenant d'opérations précédentes d'appariement. Les stratégies de combinaison orientent différents aspects du processus d'appariement, telles que le classement des candidats, l'agrégation des résultats d'appariers spécifiques et la sélection des candidats à l'appariement. La bibliothèque d'appariers proposée dans COMA est constituée de six appariers individuels, cinq appariers hybrides et un appariur de réutilisation des résultats obtenus précédemment sur des nouveaux modèles complets ou sur des fragments. Les modèles en interne sont représentés sous forme de DAG (***D****irected* ***A****cyclic* ***G****raph*). Chaque élément des schémas est identifié de manière unique par le chemin complet allant de la racine du schéma jusqu'au nœud correspondant. Ce procédé permet de capturer les contextes dans lesquels les éléments apparaissent. COMA produit des appariements de niveau élément ayant une cardinalité locale 1:1 (univoque) et une cardinalité globale m:n (multivoque).

[16] On nomme problème SAT un problème de décision visant à savoir s'il existe une solution à une série d'équations logiques données. En termes plus précis : une valuation sur un ensemble de variables propositionnelles (une valuation sur *P* est une application de *P* dans l'ensemble {0,1}, avec *P* inclus dans l'ensemble des variables propositionnelles) telle qu'une formule propositionnelle donnée soit alors logiquement vraie [Wikipedia].

### 4.3.5 Mesure de similarité générique pour l'appariement de graphe

Le dernier exemple que nous présentons n'est pas un système d'appariement mais une mesure de similarité générique dans le contexte de l'appariement de graphes. Cette mesure de similarité nous a intéressés du fait de sa généricité et de la proposition de plusieurs algorithmes d'appariement distincts dans le contexte des problèmes d'appariement de graphes.

Sorlin et Solnon ont proposé une mesure générique de distance paramétrable en fonction du type de graphes à apparier pour les différents problèmes d'appariement de graphes [Sorlin 2006] [Sorlin *et al.* 2007]. Cette mesure de similarité de deux objets est fonction de leurs caractéristiques communes sur l'ensemble de toutes leurs caractéristiques. Elle est générique dans le sens où elle modélise les mesures de distance et de similarité de graphes existantes. De plus, elle présente l'avantage d'être paramétrée par des fonctions de similarité pouvant être exprimées en fonction des connaissances propres au domaine, à l'application considérée ainsi qu'au type d'appariement recherché (univoque et/ou multivoque). De manière générale, deux fonctions de similarité permettent respectivement de calculer le score de similarité de chacun des sommets et des arcs.

Nous rappelons qu'un graphe est une structure de données utilisée notamment pour modéliser des objets en termes de composants (appelés **sommets** ou **nœuds**) et de relations binaires entre composants (appelées **arcs** ou **arêtes**). De façon plus formelle, un graphe est défini par un couple $G = (V, E)$ tel que $V$ est un ensemble fini de sommets (*vertex set*) et $E$ est un ensemble d'arcs (*edge set*) tel que $E \subseteq V \times V$. Un graphe dont les arêtes sont orientées est qualifié de **digraphe** ou **graphe orienté**. Dans un **graphe multi étiqueté**, les sommets et les arcs sont associés à plusieurs étiquettes (*labels*) décrivant leurs propriétés. Étant donnés $L_V$ (respectivement $L_E$) un ensemble d'étiquettes de sommets (respectivement d'arcs), un **graphe multi-étiqueté** est défini par un triplet $G = (V, r_V, r_E)$ tel que :

- $V$ est un ensemble de sommets ;
- $r_V \subseteq V \times L_V$ est une relation associant les sommets à leurs étiquettes, *i.e.,* $r_V$ est l'ensemble des couples $(v, l)$ tels que le sommet $v$ a pour étiquette $l$ ;
- $r_E \subseteq V \times V \times L_E$ est une relation associant les arcs à leurs étiquettes, *i.e.,* $r_E$ est l'ensemble des triplets $(v, v', l)$ tels que l'arc $(v, v')$ a pour étiquette $l$ ;
- l'ensemble $E$ des arcs est défini par $E = \{(v, v') / \exists l, (v, v', l) \in r_E\}$.

Les tuples de $r_V$ et $r_E$ constituent les caractéristiques du graphe $G$. L'ensemble $descr(G) = r_V \cup r_E$ contient toutes les caractéristiques des sommets et des arcs de $G$ et décrit entièrement le graphe $G$.

Selon Sorlin et Solnon, la similarité de deux graphes $G$ et $G'$ par rapport à un appariement $m$ de sommets et d'arcs est définie par :

$$sim_m(G, G') = \frac{f(descr(G) \cap_m descr(G')) - g(splits(m))}{f(descr(G) \cup descr(G'))}$$

où la fonction *f* pondère des caractéristiques des graphes *G* et *G'*, la fonction *splits* calcule l'ensemble des éclatements de *m*, et la fonction *g* pondère ces éclatements. Les deux fonctions *f* et *g* sont paramétrables par rapport aux besoins de l'application considérée. Le numérateur exprime le score de similarité par rapport à un appariement *m* : il est d'autant plus élevé que les graphes *G* et *G'* ont plus de caractéristiques communes. Cependant, le numérateur décroît en fonction du nombre d'éclatements. Ce score obtenu est normalisé par rapport à l'ensemble des caractéristiques des deux graphes, exprimé au dénominateur. La similarité maximale *sim(G,G')* de deux graphes *G* et *G'* est celle du meilleur appariement *m* de sommets et d'arcs, c'est-à-dire celui qui met en correspondance le plus grand nombre de caractéristiques tout en éclatant le moins possible de sommets et d'arcs.

Le calcul de la mesure de similarité est un problème NP-difficile. L'explosion combinatoire rend les méthodes de recherches complètes (basées sur une exploration exhaustive de l'espace de recherche et des méthodes de filtrage de cet espace) limitées à de très petits graphes (quelques sommets seulement) [Sorlin *et al.* 2006]. Les concepteurs de la mesure que nous venons de présenter brièvement ont proposé également trois algorithmes incomplets (ne garantissant pas l'optimalité de la solution trouvée mais ayant une complexité faiblement polynomiale) pouvant s'adapter facilement à de nombreux calculs de mesure de similarité et de distance. Nous présentons ces trois algorithmes par ordre de complexité croissante.

Le premier algorithme proposé est un **algorithme glouton** non déterministe et faiblement polynomial. Il retourne un appariement localement optimal et peut être exécuté plusieurs fois pour retourner le meilleur appariement trouvé. Il commence à partir d'un appariement vide et ajoute itérativement des couples de sommets choisis dans l'ensemble des candidats jusqu'à ce que l'ajout d'aucun autre couple ne puisse augmenter la similarité. À chaque étape, le couple à ajouter est choisi aléatoirement parmi l'ensemble des couples qui augmentent le plus la similarité.

Le second algorithme est basé sur une **recherche locale Taboue réactive** permettant d'améliorer une solution courante en explorant son voisinage : les voisins d'un appariement *m* sont les appariements obtenus en ajoutant ou en supprimant un seul couple de sommets à *m*. En démarrant à partir d'un appariement initial (en général défini à l'aide d'un algorithme glouton), une recherche locale explore l'espace de recherche en se déplaçant de voisin en voisin jusqu'à l'obtention de la solution optimale ou jusqu'à un nombre maximum d'itérations autorisé. À chaque itération, le prochain voisin à explorer est choisi selon une heuristique. Le voisin qui maximise la similarité est toujours sélectionné en premier. Une liste **Taboue** est utilisée pour mémoriser les *k* derniers mouvements réalisés afin d'interdire les mouvements inverses durant *k* itérations et ainsi ne pas rester autour d'un maximum local en réalisant toujours les mêmes mouvements.

Le dernier algorithme proposé est l'**optimisation par colonie de fourmis** (***A**nt **C**olony **O**ptimization*) [Solnon 2005] qui consiste à reformuler le problème à résoudre en un problème de recherche d'un chemin optimal dans un graphe (appelé graphe de construction) et à utiliser des fourmis artificielles pour trouver les bons chemins de ce graphe. À chaque itération de l'algorithme (appelée cycle), chaque fourmi de la colonie construit aléatoirement un chemin du graphe (une solution du problème) et de la phéromone est déposée sur les meilleurs chemins découverts lors de ce cycle. Lors des cycles suivants, les fourmis construisent de nouveaux chemins avec une probabilité dépendant de la phéromone déposée lors des cycles précédents et d'une heuristique propre au problème considéré. La colonie de fourmis converge alors peu à peu vers les meilleures solutions.

## 4.4 Conclusion

Le problème d'appariement de modèles est complexe et peut être abordé selon de nombreuses dimensions, techniques individuelles et divers algorithmes. Il ressort que l'utilisation d'une seule technique n'est pas satisfaisante pour répondre au problème d'appariement de modèles car le résultat est produit est généralement faible. Par exemple, les approches linguistiques (portant sur les noms ou les descriptions textuelles des éléments) ne permettent pas de repérer des écarts de représentation liées à leur structure dans les modèles. L'utilisation de plusieurs techniques et de plusieurs dimensions d'appariement est donc une nécessité pour traiter le problème d'appariement mais elle impose de réfléchir à la manière dont elles vont être combinées et paramétrées. Il faut garder à l'esprit que le recours à plusieurs techniques augmente les calculs et par conséquent le temps pour produire l'alignement.

Les entrées principales des systèmes d'appariement de modèles sont les modèles dont un alignement des constituants va être identifié. La représentation des modèles retenue dans beaucoup d'approches se fait sous forme de graphes et plus particulièrement sous forme de graphes acycliques orientés. Des données auxiliaires fournies en plus des modèles vont faciliter le processus d'appariement en apportant des précisions sur la sémantique des modèles (difficile à percevoir sur ces seules sources de données) et permettre dans certains cas de lever des ambiguïtés, de diriger ou d'accélérer le processus.

Les correspondances de l'alignement proposé par un système d'appariement de modèles sont focalisées sur les similarités et le plus souvent qualifiées par une relation sémantique d'équivalence ou par un score réel compris entre 0 et 1 (exprimant le degré de similarité). Les relations sémantiques sont néanmoins plus évoluées dans les techniques d'appariement sémantique (inclusion, subsumption...). Dans les techniques étudiées, nous remarquons que les écarts ne sont pas qualifiés en détail car l'accent est mis sur les « ressemblances ».

Il ressort que le problème d'appariement est difficile à traiter complètement de manière automatique par un système du fait qu'il est subjectif. L'intervention d'un utilisateur humain peut être nécessaire notamment pour désambiguïser les choix des candidats à l'appariement, choisir directement certaines correspondances ou fournir des données auxiliaires. De plus, le résultat produit est traité ensuite par un expert du domaine concerné pour en vérifier sa pertinence. Les approches proposées dans le domaine sont donc le plus souvent semi-automatiques.

# CHAPITRE 5

# La méthode d'appariement de diagrammes

# PLAN DU CHAPITRE

Notre thème de recherche se situe dans le contexte des EIAH pour la modélisation et nous nous attaquons au problème particulier d'analyse des productions de l'apprenant dans la tâche ouverte de construction d'un modèle à partir de spécifications textuelles. Nous avons pour objectif principal de définir et de mettre en place dans Diagram une méthode d'analyse des diagrammes construits par l'apprenant comme une alternative à celles existant dans un contexte d'apprentissage (cf. les approches CBM et *Curriculum* exposées dans la partie 3.3) : nous souhaitons définir une méthode ne dépendant pas des exercices ou des erreurs et ne préjugeant pas des rétroactions fournies à l'apprenant. Cette méthode doit ainsi être capable de produire des résultats en toutes circonstances. Pour cela, nous ne nous focalisons pas directement sur la notion d'erreur mais sur celle de différence. Nous considérons en effet que les choix de modélisation qu'adopte l'apprenant au cours de l'activité de modélisation, s'ils sont différents de ceux d'un diagramme de référence construit par un expert, n'expriment pas nécessairement des erreurs : l'apprenant peut avoir oublié de modéliser une partie de l'énoncé, fait un mauvais choix de modélisation ou avoir adopté une représentation différente de celle prise comme référence.

Notre méthode a pour but d'identifier et de qualifier les incohérences et les différences entre le diagramme produit par l'apprenant et un diagramme de référence exprimant une interprétation correcte de l'énoncé. Les différences relevées sont l'une des bases permettant de fournir des rétroactions pédagogiques pertinentes favorisant l'activité métacognitive de l'apprenant pour qu'il puisse réfléchir sur sa production, découvrir et corriger ses erreurs lui-même. Le système n'a pas pour objectif dans ce contexte de se substituer à l'enseignant au cours de l'activité de construction d'un diagramme UML car il ne détecte pas et ne corrige pas les erreurs. Cependant, il permet de l'assister dans son travail en encourageant l'auto-correction chez l'apprenant. Pour cela, il doit être robuste et capable de fournir ses résultats en un temps suffisamment bref pour mettre en place une interaction synchrone entre le système et l'apprenant.

Pour répondre à ces différents objectifs, nous avons étudié les approches existantes relatives à l'appariement de modèles proches des graphes (i.e. *graph-like structures*) et d'autres travaux plus spécifiques à l'interrogation, la transformation et la restructuration des modèles orientés objet. Dans le chapitre 4, nous avons exposé la terminologie, les concepts et les techniques classiques inhérents aux systèmes d'appariement de modèles. Nous nous sommes notamment inspirés des fondamentaux de ces travaux pour proposer une nouvelle méthode dédiée à l'appariement des diagrammes UML et plus particulièrement des diagrammes de classes UML de niveau analyse construits par des novices. Des aspects tels que les noms, les types, les structures (explicites et implicites), le voisinage des éléments peuvent être pris en compte lors de l'appariement de plusieurs diagrammes de classes UML. Notre méthode a pour principale particularité de s'appuyer sur la structure et la sémantique définies dans le métamodèle d'UML et les concepts orientés objet. Elle exploite également des techniques linguistiques pour comparer et mettre en correspondance les espaces de nommage des éléments UML représentés sous forme de chaînes de caractères. Afin de qualifier l'alignement produit par notre méthode (i.e. les correspondances identifiées entre les modèles), nous proposons une taxonomie de différences pouvant s'appliquer à différents types de modèles et expliquant les écarts relatifs essentiellement aux structures des constituants des modèles.

Notre contexte d'apprentissage et les besoins liés à la production de rétroactions synchrones imposent des spécificités à notre problème d'appariement. Elles orientent et contraignent les choix et le fonctionnement de la méthode d'appariement à mettre en place.

Dans la suite du chapitre, nous détaillons tout d'abord les spécificités de notre problème d'appariement dans le contexte de l'apprentissage des concepts orientés objet, puis le fonctionnement général retenu dans notre méthode. Nous détaillons ensuite la représentation et le prétraitement des données en entrée (les diagrammes de classes UML), la mesure de similarité et le processus d'appariement. Enfin, après avoir précisé le résultat produit (les différences), nous concluons par le paramétrage de la méthode et une évaluation de sa complexité.

## 5.1 Particularités de notre contexte d'apprentissage

Notre étude des environnements d'apprentissage de la modélisation (cf. chapitre 3) a montré que l'analyse de la production de l'apprenant et l'identification de ses erreurs commises lors d'une activité de construction d'un modèle à partir d'un énoncé textuel ne peut être réalisée qu'en ayant au moins un diagramme « correct » comme référence pour ce même énoncé. La confrontation de ces modèles de même nature peut être vue comme un problème d'appariement de modèles dont l'objectif consiste à identifier et à qualifier les correspondances (et les différences) entre les éléments des diagrammes les uns par rapport aux autres. Le choix d'un ensemble de techniques d'appariement pour traiter ce problème est dirigé par les modèles à apparier, le contexte et le domaine d'application. Nous nous penchons plus particulièrement dans cette partie sur les spécificités propres au domaine d'apprentissage (les écarts de représentation et la notion de modèle de référence) puis sur les informations externes aux modèles que nous avons à notre disposition.

### *5.1.1 Des diagrammes très différents élaborés par l'apprenant*

Nous nous intéressons ici au modèle élaboré par l'apprenant dans le contexte d'apprentissage des concepts orientés objet et notamment dans l'environnement Diagram. Lorsqu'un modèle est élaboré par un apprenant, le sens véhiculé par son modèle peut être très éloigné des concepts exprimés par l'énoncé de l'exercice de modélisation. Si la structure des éléments représentant des concepts est altérée alors le sens du modèle ne sera pas conforme à l'énoncé proposé. De plus, les modèles produits peuvent être à la fois sous-spécifiés, sur-spécifiés et altérés. Enfin dans Diagram, l'activité de modélisation est moins contrainte que dans les environnements reposant sur CBM et dans DesignFirst-ITS (cf. partie 3.3) : dans Diagram, les noms employés dans les diagrammes sont choisis librement par l'apprenant et certains concepts peuvent être implicites dans les énoncés des exercices. Nous allons montrer en quoi ces différents aspects complexifient le problème d'appariement et la manière dont ils peuvent être traités.

#### 5.1.1.1 L'altération des relations structurantes

Dans les modèles hiérarchiques, les relations orientées de classification (*is-a*, ou relation de taxonomie ou d'héritage en MOO) et de propriété structurent les concepts sous forme d'arborescence. Les approches classiques d'appariement de modèles représentant les modèles sous forme de graphes traitent ces relations comme des arcs pris en compte lors de l'évaluation des similarités des nœuds liés et de leur voisinage (cf. les techniques individuelles de niveau-structure présentées dans la partie 4.2.2). Dans les contextes d'application courants des approches de *schema matching* ou d'*ontology matching* telles que l'intégration et la fusion de données, les liens entretenus entre les nœuds sont toujours corrects car les modèles à apparier ne présentent pas d'erreurs : par exemple, les schémas relationnels de deux sites web sont corrects par nature même si leurs informations ne sont pas organisées de la même manière. Dans

notre contexte lié à l'apprentissage humain, les relations représentées par l'apprenant (et les éléments de manière générale) sont très souvent incorrectes (e.g. orientation inversée, substitution d'une relation par une autre d'un autre type, relation absente du modèle...). Une simple inversion de l'orientation d'une relation ou l'utilisation d'un autre type de lien sémantique modifie fortement le sens des nœuds liés directement ou indirectement (leur voisinage proche ou éloigné). Ces modifications de sens influencent le choix des candidats d'une méthode d'appariement. L'altération de la structure et du sens en découlant dans le modèle de l'apprenant induisent que les relations (graphiques des diagrammes) doivent elles aussi être comparées suivant des critères qui leur sont propres et alignées tout comme les éléments qu'elles lient dans les diagrammes de classes UML.

Nous avons proposé trois interprétations possibles des diagrammes de classes UML sous forme de graphes dans la partie 1.4.2. Il est important de remarquer que les relations graphiques d'un diagramme de classes ne sont pas transcrites de la même manière (cf. figure 36) : notamment dans la dernière représentation sous forme de graphe UML, une association est un nœud du graphe alors qu'une relation d'héritage est un arc entre des classes (cf. les deux premières représentations de la figure 36). Cette différence pose un problème pour les comparer et les aligner étant donné que le processus d'appariement est mené classiquement sur les nœuds en prenant en compte les caractéristiques et les liens avec les autres nœuds. L'appariement d'un nœud et d'un arc n'est pas concevable dans les approches classiques d'appariement de graphes. Pour pallier ce problème, il est nécessaire de représenter la relation d'héritage sous forme d'un nœud du graphe dans notre cas. Une représentation arborescente du diagramme de classes est une première solution pour répondre à ce problème : une relation d'héritage est un nœud lié à la classe fille par une relation de propriété et à la classe mère par un attribut de l'héritage noté *general* (cf. troisième représentation de la figure 36). Une autre solution consiste à considérer que l'héritage « connaît » explicitement la classe mère et la classe fille qu'elle connecte en représentant deux liens orientés vers les classes liées (cf. les arcs notés *superclass* et *subclass* de la quatrième représentation de la figure 36).

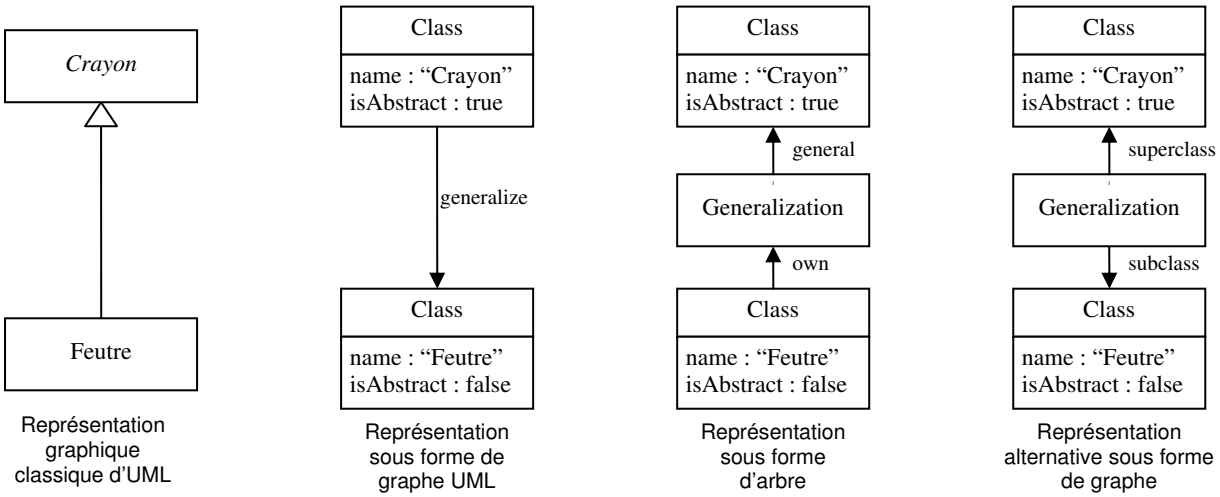



**Figure 36 : Exemples de représentation de la relation d'héritage**

Nous retenons la troisième représentation sous forme d'arborescence pour représenter les diagrammes de classes UML en entrée de notre méthode d'appariement. Cette représentation est celle utilisée dans le standard XMI 2 défini par l'OMG pour échanger et manipuler les diagrammes UML. Les éditeurs UML implantant une représentation des diagrammes UML conforme au métamodèle actuel d'UML 2.x la propose très souvent pour exporter les diagrammes construits.

#### 5.1.1.2 Des diagrammes sous-spécifiés, sur-spécifiés et altérés

Dans le contexte particulier de l'apprentissage de la modélisation par des novices, les concepts représentés sous forme d'éléments UML peuvent être altérés syntaxiquement, structurellement et sémantiquement. L'utilisation différente des éléments UML (représentation d'un type de relation à la place d'un autre, d'un attribut à la place d'une classe ou inversement...), la non représentation de certains éléments ou l'ajout d'autres éléments corrects ou non dans le modèle sont des exemples de différences courantes qui complexifient[17] le problème d'appariement.

Des parties de l'énoncé peuvent ne pas avoir été modélisées par l'apprenant dans son diagramme : le diagramme est alors incomplet (**sous-spécifié**). Dans ce cas, l'apprenant peut avoir « oublié » ou omis de représenter seulement quelques éléments ou tout une partie du diagramme de classes UML. Ces omissions ont des conséquences à différents niveaux dans le diagramme. Par exemple, l'absence d'une classe implique que les attributs, les opérations qu'elle encapsule et les relations qui les relient à d'autres classes du diagramme sont également absents (ou représentés à un autre endroit et/ou sous une autre « forme »). La non-représentation d'un attribut a une conséquence locale au niveau du sens de la classe qui devait le contenir (i.e. une propriété est absente). De plus, si l'attribut omis devait être hérité par les classes filles de la classe propriétaire alors le sens véhiculé par ces classes sera également incomplet. À l'échelle des structures de plus haut niveau de granularité telles que des hiérarchies de classes (des classes reliées par des relations d'héritage), la non-représentation d'une partie ou de l'intégralité d'une hiérarchie affecte l'ensemble des classes, des attributs, des opérations et des relations constituant cette hiérarchie mais également les autres éléments liés aux constituants de cette hiérarchie.

Des parties du diagramme allant d'un simple élément à des ensembles d'éléments liés les uns aux autres peuvent être superflus. Ces éléments ajoutés au diagramme apportent des informations supplémentaires pouvant être alternatives ou non pertinentes. La **sur-spécification** est le problème inverse de la sous-spécification. Elle a des conséquences à différents niveaux de granularité du modèle : par exemple, l'ajout d'une classe par l'apprenant a des effets sur la structure interne de la classe elle-même et externe par rapport aux autres éléments de son voisinage : les attributs, les opérations encapsulés et les relations les liant à d'autres classes sont insérés localement à cette classe. Un élément ajouté au modèle n'est pas nécessairement une erreur commise par l'apprenant car il peut être relatif à un ajout d'informations supplémentaires : par exemple, la représentation d'un attribut dérivant d'un autre attribut dans une classe constitue une information utile dans certains cas de modélisation. Cependant, l'ajout d'une relation d'héritage entre deux classes, n'admettant pas de lien de classification entre les concepts qu'elles représentent, est une erreur qui modifie le sens des classes et des autres éléments appartenant à leur voisinage dans le reste du diagramme (i.e. propagation des propriétés et du comportement des classes par héritage).

Des parties du diagramme construit par l'apprenant peuvent être modifiés ou erronées : le diagramme est « **altéré** », c'est-à-dire qu'un ou plusieurs éléments ou leurs relations avec le reste du diagramme peuvent être altérés aussi bien au niveau de leur syntaxe, de leur structure ou de leur sémantique. La représentation d'un attribut dans une classe qui ne devrait pas l'encapsuler ou la représentation d'un élément d'un type par un autre élément d'un autre type

[17] Intuitivement, une tâche d'appariement avec des schémas de même taille est plus difficile à réaliser si la similarité entre eux baisse [Do *et al.* 2002].

sont des exemples d'altérations simples. Dans le premier exemple, localement à la classe qui aurait dû contenir l'attribut, cet attribut y est absent alors que localement à la classe où il est représenté, il est superflu. À l'échelle des modèles à apparier, l'attribut a été déplacé par l'apprenant d'une classe vers une autre. Des altérations sont plus complexes : des éclatements, des fusions ou encore des regroupements de concepts. Par exemple, un concept de l'énoncé peut avoir été représenté sous forme de plusieurs classes distinctes alors qu'une seule classe est nécessaire. Ce genre d'altération a des effets locaux sur les classes représentées (par exemple, des attributs encapsulés en doublon, des noms relativement similaires ou complémentaires) et au niveau des liens qu'elles entretiennent avec les autres éléments du modèle (par exemple, des liens redondants ou répartis entre les différentes classes représentant le même concept et le reste du diagramme). À chaque niveau de granularité, des altérations peuvent exister et dépendre les unes des autres. Si une classe racine d'une arborescence de classes encapsulant plusieurs attributs a été omise dans le diagramme, l'apprenant a pu représenter les attributs et les relations (devant la relier à d'autres classes du modèle) au niveau de n'importe quelle classe fille de l'arborescence. Dans cet exemple, plusieurs attributs et relations peuvent être redondants et ne pas constituer une représentation fidèle de l'énoncé ou conforme au paradigme objet.

Notre méthode doit être capable de fournir un résultat en toute circonstance en gérant le fait qu'un même diagramme peut être à la fois sous-spécifié, sur-spécifié et « altéré ». La variabilité d'interprétation et de représentation des énoncés par les apprenants ainsi que les erreurs qu'ils commettent en les modélisant sous forme d'un diagramme de classes UML imposent de comparer les modèles à différents niveaux de granularité en prenant un maximum de dimensions décrivant leurs constituants. De plus, la mise en correspondance, pour être pertinente dans notre contexte, nécessite de qualifier des appariements univoques et multivoques dans les cas les plus complexes (par exemple, lorsque le modèle élaboré par l'apprenant est altéré au niveau de sa structure générale). Il est donc nécessaire de mettre en place une mesure de similarité **multivoque tolérante aux erreurs** (cf. partie 4.1.4) lors du processus d'appariement.

### *5.1.2 La référence construite par l'enseignant*

Dans le cadre de la modélisation orientée objet, plusieurs diagrammes modélisent correctement un même énoncé : ils peuvent être structurellement différents mais équivalents sémantiquement (notamment lors de l'utilisation de *patterns* d'analyse ou de conception). Un unique diagramme de référence ne permet pas de déduire toutes les alternatives de représentation d'un énoncé textuel. L'appariement entre le diagramme élaboré par l'apprenant et une référence correcte élaborée par un expert (par exemple l'enseignant proposant l'exercice de modélisation) peut ne pas être optimal dans les cas où ces diagrammes représentent des solutions alternatives.

Une première solution pour répondre partiellement à ce problème est de repérer des *patterns* d'analyse et/ou de conception dans les diagrammes à comparer car elles permettent de déduire des constructions équivalentes sémantiquement mais différentes structurellement. Néanmoins, toutes les équivalences ne sont pas déductibles de cette manière. Une autre solution pour répondre à ce problème est de confronter le diagramme de l'apprenant à plusieurs diagrammes de référence, qu'ils soient corrects ou erronés et complets ou partiels. La prise en compte de plusieurs références lors de l'analyse d'un diagramme construit par l'apprenant apporte plusieurs difficultés supplémentaires. La première est la suivante : chaque confrontation des éléments du diagramme de l'apprenant avec ceux d'un fragment ou d'un diagramme complet est un problème d'appariement à part entière et distinct. Ainsi, le temps et le nombre de calculs

nécessaires pour traiter ces problèmes sont dépendants du nombre de modèles à comparer. La seconde difficulté est de répondre à la question suivante :

*Comment décrire et qualifier les diagrammes les uns par rapport aux autres ?*

Plusieurs cas se présentent : si les diagrammes sont exclusifs structurellement mais « sémantiquement » équivalents (c'est rarement le cas pour des diagrammes complets car des *patterns* sont appliquées en général seulement sur un fragment d'un diagramme), il n'y a pas difficulté majeure car il suffit de sélectionner l'alternative s'appariant le mieux au diagramme défini par l'apprenant (celle présentant le plus de similarités). Dans les autres cas de figure tels que des diagrammes admettant des parties en commun ou représentant un sous-ensemble alternatif d'un autre diagramme (i.e. un problème d'appariement de sous-graphes), il est difficile de qualifier des relations explicites et précises entre les références. Cette tâche est également un problème d'appariement en soi qui ne peut être résolu que manuellement par un expert du domaine ou de manière semi-automatique.

Dans notre contexte d'apprentissage des concepts de base de la MOO, les énoncés des exercices proposés aux novices conduisent le plus souvent à l'élaboration de diagrammes de taille faible ou moyenne (i.e. aux alentours d'une soixantaine d'éléments pour les plus grands issus de notre corpus) et ayant rarement une complexité pouvant justifier l'emploi de *patterns* d'analyse ou de conception. Peu d'alternatives très différentes structurellement existent dans ces conditions. Dès lors que les exercices n'admettent pas plusieurs solutions totalement exclusives, il est possible d'utiliser une référence unique définie par l'enseignant lors du processus d'appariement. Une référence permet d'identifier quelques constructions alternatives en se focalisant sur les propriétés des structures (par exemple les hiérarchies de classes). Nous pensons que l'alignement du diagramme de l'apprenant et d'une référence est pertinent pour orienter la réflexion de l'apprenant novice sur sa propre production (cf. chapitre 6) mais qu'il n'est pas toujours suffisant pour indiquer explicitement des erreurs à l'apprenant car sa proposition pourrait être une alternative non déductible de la référence (certaines erreurs telles que l'inversion du sens d'un héritage peuvent être détectées avec une seule référence).

### 5.1.3 Les informations auxiliaires et l'utilisation du résultat produit

La plupart des appariers ne comptent pas seulement sur les modèles en entrée mais également sur des ressources ou des informations auxiliaires (externes) comme par exemple des dictionnaires, des thésaurus, des choix d'appariement précédents, des saisies de l'utilisateur, des schémas globaux...

Dans l'environnement Diagram (cf. partie 2.4), en dehors des diagrammes à apparier et de l'énoncé à partir desquels les diagrammes ont été créés, notre méthode ne dispose pas d'informations auxiliaires à prendre en compte avant, pendant et à l'issue du processus d'appariement. Les différents utilisateurs ne peuvent pas apporter d'autres informations. L'apprenant construit son diagramme par rapport à l'énoncé et n'a pas connaissance des solutions valides construites par l'enseignant. Ce dernier définit l'énoncé et un diagramme de référence avant que l'activité de modélisation ne débute. Lors de l'activité de modélisation, des rétroactions pédagogiques sont fournies en synchrone à partir des résultats de la méthode d'appariement (l'enseignant n'intervient pas sur l'environnement lors de l'activité).

Il n'y a pas de possibilités de consultation des utilisateurs dans le contexte d'application de notre méthode d'appariement. Le résultat du système d'appariement doit donc être déterminé de manière automatique et suffisamment

rapide pour des besoins de rétroactions synchrones sans bénéficier de l'intervention d'un acteur humain pour lever les ambiguïtés relatives aux candidats d'appariement. Le résultat de notre système est vu comme une solution (cf. partie 4.2.1) au problème d'appariement posé entre le diagramme de l'apprenant et celui de référence et non pas comme un ensemble d'indications comme c'est souvent le cas dans beaucoup de systèmes d'appariement de modèles.

## 5.2 Introduction de motifs structurels caractéristiques

Les techniques d'appariement de modèles sous forme de graphes se focalisent essentiellement sur les sommets et les arcs mais elles ne prennent pas directement en compte les structures plus complexes pouvant agencer les éléments à un niveau plus général. Or, le problème de qualification de différences de déplacement d'éléments ne peut pas être géré complètement à l'échelle locale des éléments. Nous avons donc choisi différents niveaux de granularité d'appariement dans notre méthode pour qualifier des différences plus générales, prendre en compte des constructions alternatives inhérentes à la structure des diagrammes et enfin orienter le processus d'appariement de notre méthode. Ces niveaux de granularité sont exprimés dans notre proposition par des structures nommées **motifs structurels caractéristiques**.

### *5.2.1 Des besoins d'interrogation et de structuration des diagrammes*

La comparaison d'un élément d'un modèle avec un élément d'un autre modèle tient compte de leur description locale et des liens qu'ils entretiennent avec leur voisinage proche (leur conteneur et les éléments directement liés à eux). Or dans certains cas, un élément peut ne pas avoir été représenté au « bon endroit » d'un diagramme (cf. partie 5.1.1). Pour pouvoir identifier et qualifier ce type de différences, il est nécessaire d'avoir des connaissances plus poussées sur la structure et la sémantique des diagrammes. Un dispositif approprié d'interrogation des diagrammes peut permettre de connaître ces informations. La mise en place de dispositifs d'interrogation des diagrammes UML et de leurs éléments a été traitée dans les travaux de Raimbault *et al.* en codant les différentes notations UML dans une ontologie et en transformant ensuite automatiquement les diagrammes UML sous forme de graphes conceptuels[18] [Raimbault *et al.* 2006]. Les auteurs proposent une interrogation complète qui permet par exemple de connaître « les opérations publiques d'une classe donnée » ou de chercher directement « les sous-classes non abstraites d'une certaine classe ». Dans leur cas, la vérification de la sémantique d'un diagramme est exprimée par des spécifications orientées objet et des spécifications du domaine sous forme de contraintes positives ou négatives.

L'identification de structures complexes dans les diagrammes UML est réalisée et est utilisée dans le cadre de la réutilisation de composants également appelés motifs de conception (*design patterns*) à tout niveau d'abstraction. Dans ce contexte, Khayati a défini une technique structurelle externe de recherche de composants dans un diagramme de classes UML en se basant sur un mécanisme d'appariement de spécifications [Khayati 2005]. Comme dans notre contexte, les spécifications sources et cibles de l'appariement sont des diagrammes de classes UML. Cependant, dans ces travaux, les spécifications comparées représentent respectivement le composant réutilisable et les besoins d'un ingénieur d'application désirant retrouver un composant répondant à ses critères de recherche. Cette technique structurelle s'intéresse aux signatures des composants et à la composition interne des composants en termes de classes,

[18] Les graphes conceptuels permettent de représenter, de raisonner et de visualiser des connaissances.

d'associations, de compositions, d'héritages, etc. Khayati a utilisé le formalisme de la logique du premier ordre pour représenter les spécifications car elle offre l'avantage de permettre la description d'un diagramme de classes sous la forme de formules logiques et facilite des opérations de recherche.

La recherche de motifs de conception n'est pas adaptée à notre problème d'appariement car notre objectif n'est pas d'améliorer un diagramme déjà correct en y recherchant des patrons prédéfinis. Nous souhaitons rechercher et qualifier des différences dans le diagramme de l'apprenant (source) par rapport à un diagramme de référence (cible). Toutefois, la technique structurelle avancée par Khayati peut être pertinente en se focalisant sur d'autres types de constructions moins générales mais communes à tous les diagrammes comparés. Dans le cadre de la modélisation orientée objet, les diagrammes sont fortement structurés sous forme de hiérarchies. Dans les diagrammes de classes UML, les objets et classes sont ordonnés sont forme de hiérarchies dont les deux plus importantes sont la **« sorte de »** (ou hiérarchie d'héritages) structurant les classes autour des héritages et la **« partie de »** (ou hiérarchie d'agrégations ou de compositions) structurant les objets autour de la notion d'agrégation. Les hiérarchies, du fait de leur sémantique et de leur niveau de granularité, peuvent être interrogées pour connaître les liens qui existent entre les différents éléments les constituant. De plus, ces hiérarchies possèdent des propriétés intéressantes (la propagation et la dérivation) qui peuvent être utilisées pour déduire des constructions alternatives à un diagramme donné en référence. Enfin un déplacement d'élément au sein d'une hiérarchie a un sens qui peut être qualifié. Nous nous focalisons donc sur les concepts de hiérarchie et d'interrogation dans notre méthode d'appariement.

### *5.2.2 Motifs simples et complexes*

Nous avons introduit des structures caractéristiques appelées motifs qui correspondent aux différents niveaux de granularité d'un modèle. Nous différencions deux principaux niveaux de granularité : les **motifs simples** et les **motifs complexes**. Les motifs simples correspondent aux éléments représentables dans les modèles comparés. Ces motifs simples sont organisés et structurés par des motifs complexes, c'est-à-dire des ensembles d'éléments formant des constructions particulières et possédant une sémantique bien définie. Dans le cas des diagrammes de classes UML, les principaux motifs simples et complexes identifiables sont exposés dans la figure 37.

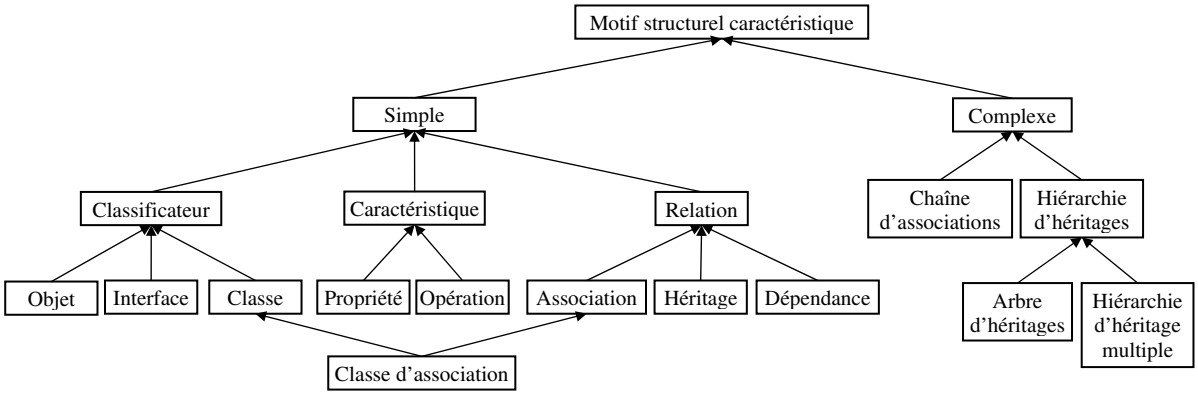



**Figure 37 : Motifs structurels caractéristiques des diagrammes de classes UML 2**

Les motifs simples des diagrammes de classes sont ceux définis par l'OMG dans le métamodèle d'UML 2.x. Trois types majeurs de motifs simples existent dans les diagrammes UML : les classificateurs, les relations et les

caractéristiques. Les classificateurs sont liés aux relations et tous les deux possèdent des caractéristiques qui leur sont propres (pour les relations, seules les associations possèdent des caractéristiques).

Dans les diagrammes UML, les motifs complexes sont centrés sur la notion de hiérarchie intrinsèque à la modélisation orientée objet. Les motifs complexes, les **chaînes d'associations** et les **hiérarchies d'héritages**, sont construites autour des classificateurs et d'un type de relation. Ces relations sont les associations et les héritages qui renvoient aux concepts des hiérarchies « partie de » et « sorte de ». Notre terminologie pour nommer les motifs complexes est focalisée sur le type de relation agençant les classificateurs car nous considérons que ce sont les relations qui donnent leur sens aux hiérarchies en liant et orientant les classificateurs les composant. Un motif complexe minimal est constitué à partir d'au moins trois motifs simples correspondant à deux classificateurs et une relation.

Les relations d'héritage agencent les classes sous forme d'arbres d'héritages où la classe mère, dont toutes les classes héritent directement ou indirectement, est nommée racine. Les classes les plus spécialisées (sans classe fille) sont les feuilles de l'arbre d'héritages. Chaque chemin allant d'une feuille à la racine constitue une branche de l'arbre. L'héritage multiple donne lieu à des hiérarchies d'héritage multiple composées de plusieurs arbres d'héritages distincts ayant une classe commune. Cette classe joue le rôle de pont entre les deux arbres d'héritages, en héritant d'une partie des propriétés et du comportement des éléments constituant chacun des arbres.

Les relations d'association agencent les classes sous forme de chaînes d'associations où les classes liées à une seule autre classe sont les extrémités des chaînes. Différents chemins peuvent lier une extrémité de la chaîne à une autre extrémité. Nous ne différencions pas les relations d'association, d'agrégation et de composition car conceptuellement elles représentent des relations d'association plus ou moins spécialisées. De plus, les relations d'agrégation et de composition peuvent être représentées par des relations d'association sans précision de leur type d'agrégation dans les diagrammes de niveau analyse en UML.

### *5.2.3 Schématisation des diagrammes de classes UML en motifs*

L'identification de motifs complexes dans un diagramme permet d'avoir une vue plus générale des liens entretenus entre les éléments des diagrammes. Nous nommons **schématisation** l'identification des motifs dans les diagrammes car graphiquement, les motifs complexes donnent une vue schématique du diagramme en encapsulant et en masquant les éléments qui les composent.

Un exemple de schématisation en motifs est donné dans la figure 38 pour le diagramme de classes détaillé de l'exercice « Stylo et Feutre » dont nous avons présenté différentes représentations sous forme de graphe dans la partie 1.4. Les motifs simples sont ici sept classes, sept relations et les différentes caractéristiques encapsulées dans les classes et les relations d'association. Ces motifs simples sont regroupés en trois motifs complexes : un arbre d'héritages (le triangle vert) et deux chaînes d'associations (les rectangles). Nous pouvons remarquer que l'arbre d'héritages est la structure centrale de ce diagramme autour de laquelle tous les éléments sont organisés.

L'intérêt majeur de la schématisation des diagrammes est de pouvoir ensuite focaliser l'appariement sur les structures les plus complexes qui sont moins nombreuses que les motifs simples : il est plus simple de confronter et

d'apparier quelques motifs complexes et ensuite de traiter les motifs simples les constituant. Tous les motifs peuvent être interrogés pour connaître leur vision interne et externe des liens qu'ils entretiennent avec les autres motifs. L'interrogation des diagrammes est ainsi mieux dirigée par l'introduction des motifs complexes : l'interrogation d'une hiérarchie d'héritages facilite la connaissance d'une part des motifs pères, frères, fils d'un motif précis et d'autre part des motifs (propriétés et opérations contenus dans les motifs parents) dont il hérite ou qui lui sont liés directement ou indirectement (relations liées à d'autres motifs apparaissant dans la hiérarchie).

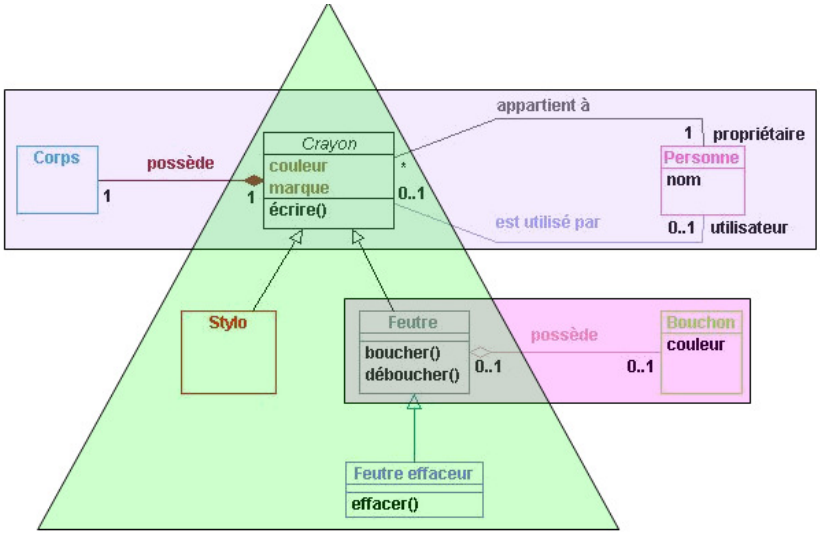



**Figure 38 : Diagramme de classes UML détaillé de l'exercice « Stylo et Feutre » schématisé en motifs**

### 5.2.4 Connaissances spécifiques des motifs

Les connaissances propres aux motifs découlent des relations de **contenance**, de **typage** et de **liaison** avec le reste des autres motifs du diagramme. Tous les motifs appartiennent à un diagramme et le diagramme connaît tous les motifs le constituant. Les motifs complexes connaissent les différents classificateurs et les relations les constituant, les autres motifs complexes liés et par conséquent les relations jouant le rôle de pont entre eux. Les motifs simples connaissent les motifs complexes dans lesquels ils interviennent, les relations qui les lient aux autres motifs simples et enfin les motifs simples qu'ils contiennent (des caractéristiques).

Pour pouvoir déduire des solutions alternatives au modèle idéal proposé par l'enseignant pour un énoncé textuel, nous exploitons des méta-connaissances issues du paradigme orienté objet. Ces méta-connaissances découlent notamment des deux types des motifs complexes que nous avons identifiés précédemment pour schématiser les diagrammes de classes et sont inhérentes à la **propagation,** la **dérivation** et la **relaxation**. Les aspects sémantiques de la généralisation et les techniques de relaxation sont déjà utilisées par Khayati dans sa technique structurelle externe de recherche de composants dans un diagramme de classes UML [Khayati 2005].

La généralisation a une sémantique plus large que le simple fait de relier deux classes car elle implique notamment la propagation des propriétés (relations, attributs, opérations) de la classe plus générale vers les classes spécialisées. La propagation descendante (des classes mères vers leurs filles) est un concept majeur de la modélisation orientée objet qui permet de structurer et d'éviter des redondances au sein des diagrammes. Les mauvais usages de la

propagation peuvent être repérés en comparant les hiérarchies d'héritages censées représenter les mêmes concepts. Notamment, en interrogeant les hiérarchies d'héritages, il est possible de repérer des problèmes de **sous-spécification** par propagation descendante (certaines des caractéristiques d'un élément ont été représentées « plus bas » dans la hiérarchie), de **sur-spécification** par propagation ascendante (certaines des caractéristiques d'un élément ont été représentées « plus haut » dans la hiérarchie) et de redondance (certaines des caractéristiques d'un élément ont été représentées plusieurs fois dans la hiérarchie).

Le **mécanisme de relaxation** permet à un processus d'appariement de ne pas retourner un résultat du type « tout ou rien » en retrouvant des éléments qui se ressemblent à quelques détails près. La relaxation est une base de notre proposition dans les motifs complexes correspondant aux chaînes d'associations et s'applique sur les associations (association ↔ agrégation ↔ composition) de manière descendante ou ascendante. La relaxation peut également s'appliquer sur le type d'un paramètre par généralisation ou spécialisation, sur une multiplicité monovaluée (1 ↔ 0..1) et sur une multiplicité multivaluée (1..* ↔ 0..*).

La dernière méta-connaissance concerne la **restructuration des éléments**. Il est possible en effet de représenter certains concepts sous la forme de différents éléments ou groupe d'éléments équivalents. Le concept de classe d'association en UML (cf. la figure 10 du chapitre 1.3.3.2) est représentable sous deux formes différentes qui sont équivalentes sémantiquement dans la majorité des cas.

## 5.3 Fonctionnement général de la méthode d'appariement

Nous avons présenté la manière dont les diagrammes de classes UML en entrée de notre système d'appariement peuvent être exprimés sous forme de graphes (cf. partie 1.4) et les informations pouvant être apportées par leur schématisation en motifs lors d'un processus d'appariement (cf. partie 5.2). Nous allons maintenant exposer notre méthode d'appariement qui à partir des diagrammes de classes UML en entrée et plus particulièrement de leurs motifs complexes et simples va évaluer leurs similarités et différences pour proposer en sortie un alignement des motifs. Cette méthode d'appariement nommée ACDC (***A**utomatic **C**lass **D**iagrams **C**omparator*) est adaptée aux spécificités propres des diagrammes de classes UML de niveau analyse et à leur élaboration dans le contexte d'apprentissage des concepts fondamentaux de la modélisation orientée objet par des novices. Elle compare (mesure les similarités et les différences) et apparie les motifs structurels de plusieurs diagrammes de classes UML fournis en entrée en se focalisant sur de nombreuses dimensions descriptives des éléments UML et de leur organisation dans les diagrammes de classes UML. Les dimensions prises en compte et leurs poids respectifs sont tous paramétrables dans notre méthode et peuvent être adaptés en fonction des besoins.

Nous avons défini ACDC sous forme d'un système d'appariement hybride paramétrable qui prend en entrée des diagrammes de classes UML et retourne en sortie un *mapping* de leurs motifs exprimé à différents niveaux de granularité. Selon la classification des techniques d'appariement d'Euzenat et Shvaiko (cf. partie 4.2.2), notre approche combine des techniques individuelles aussi bien syntaxiques (basées sur les chaînes de caractères des espaces de nommage et des contraintes de type) que structurelles (basées sur les propriétés et l'agencement des motifs). Notre approche se focalise à la fois sur les éléments et les structures des modèles à apparier (techniques de niveau élément et structure) grâce à la schématisation en motifs des diagrammes analysés.

Les diagrammes de classes UML à notre disposition sont celui construit par un apprenant à partir d'un énoncé textuel, noté $DC_S$ (pour diagramme de classes source) ainsi qu'un diagramme de classes UML de référence (un diagramme défini par l'enseignant pour le même énoncé), notés $DC_C$ (pour diagramme de classes cible).

ACDC produit comme résultat non seulement un *mapping* des éléments correspondants les uns aux autres entre le $DC_S$ et le $DC_C$ mais il qualifie également les différences entre ces éléments étant donné qu'ils peuvent avoir été altérés au niveau de leur syntaxe, leur structure et leur sémantique. La plupart des approches d'appariement sont restreintes à la cardinalité locale 1:1 par la sélection comme candidat d'appariement pour un élément d'un schéma de l'élément le plus similaire d'un autre schéma [Do *et al.* 2002]. Dans notre cas, les appariements relevés sont multivoques à toutes les échelles et ont des cardinalités locales 1:1, 1:n, n:1 et globales n:m (en combinant les appariements de cardinalités locales 1:1, 1:n et n:1). Les appariements du *mapping* final sont qualifiés par un score de similarité et des différences. Ces différences décrivant les appariements sont exprimés du $DC_S$ vers le $DC_C$ car notre problème consiste à proposer une analyse des productions de l'apprenant.

Sous forme de schéma, le processus d'appariement d'ACDC est le suivant :

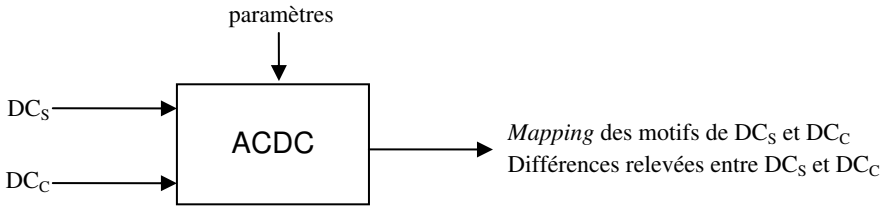



**Figure 39 : Le processus d'appariement d'ACDC**

Notre méthode d'appariement suit quatre étapes séquentielles. Les deux premières sont des étapes de prétraitement des modèles en entrée. Les suivantes constituent le processus d'appariement et de présentation de son résultat :

- **Encodage des diagrammes sous forme d'arbres XMI** : en premier lieu, chaque diagramme de classes UML (source et cible) en entrée est encodé dans le format XMI 2.1 (si ce n'était pas son format d'origine). L'utilisation de ce formalisme, standard dans la communauté UML, facilite notamment la sauvegarde, le chargement et la réutilisation des diagrammes de classes UML. Un diagramme est représenté sous la forme d'un arbre (un graphe connexe et sans cycle) où la racine de l'arbre est le modèle et où chaque élément UML (modèle, classificateur, relation, caractéristique, commentaire) représente un nœud de l'arbre. Les nœuds dans cette représentation sont liés entre eux par des relations sémantiques de propriété.
- **Schématisation des arbres XMI en motifs** : dans un second temps, les arbres sont parcourus de manière indépendante et des motifs structurels caractéristiques en sont extraits. Ces motifs permettent de faire ressortir explicitement les relations implicites relatives à la structure des modèles à apparier (contenu hérité, contexte et voisinage). Des relations de contenance, de typage, de liaison, de filiation sont relevées à l'issue du processus de schématisation entre les motifs complexes et simples (et réciproquement), et entre les motifs simples les uns par rapport aux autres.
- **Évaluation des similarités et des différences locales à chaque couple de motifs** : cette phase permet d'évaluer les similarités et les différences à différentes échelles des diagrammes. Les motifs constituant le couple de diagrammes ($DC_S$, $DC_C$) sont comparés par paires et par type. Localement à chaque couple de

motifs, un score de similarité est calculé et des appariements locaux relatifs aux motifs contenus, conteneurs, et liés des motifs confrontés sont étudiés et qualifiés par des différences univoques. À l'issue de cette phase, un pré-appariement univoque du couple ($DC_S$, $DC_C$) est obtenu. De plus, chaque couple de motifs comparés a également une vision locale de ses appariements locaux.

- **Choix de l'appariement des motifs et des différences :** le *mapping* final des motifs constituants les diagrammes du couple ($DC_S$, $DC_C$) est déterminé de manière gloutonne à l'aide de deux appariers spécifiques appelés séquentiellement et se chargeant respectivement de la détermination d'un appariement des motifs complexes et d'un appariement des motifs simples. Pour cela, ils déterminent et parcourent des listes de différences issues du pré-appariement univoque du couple ($DC_S$, $DC_C$) et des appariements locaux des motifs triés par score et par type. Les appariers ajoutent au fur et à mesure les appariements qui admettent une « bonne similarité » et sont cohérents avec l'appariement courant. Les appariements multivoques (cardinalités 1:n, n:1 et m:n) et hétérogènes (des éléments de natures différentes) sont étudiés par les appariers et sont ajoutés également au fur et à mesure au *mapping*.

La figure 40 illustre les trois dernières étapes de la méthode d'appariement ACDC prenant en entrée les diagrammes $DC_S$ (ici le diagramme de l'apprenant) et $DC_C$ (ici le diagramme de référence) sous forme d'arbres XMI et retournant en sortie un *mapping* de leurs motifs appariés et un listage des différences relevées.

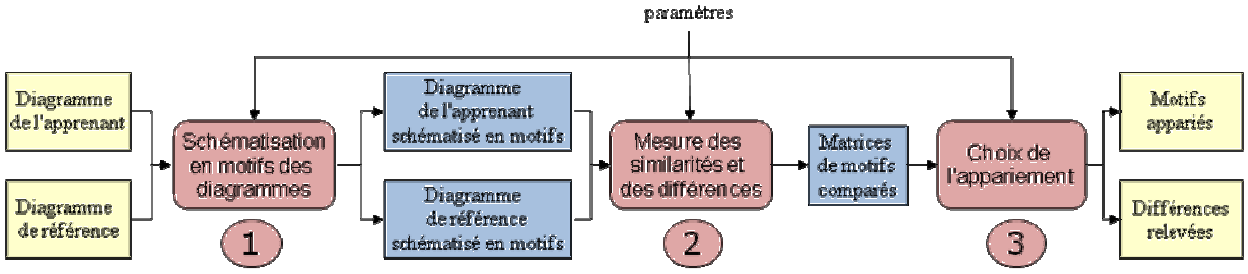



**Figure 40 : Étapes séquentielles de la méthode d'appariement ACDC**

## 5.4 Mesure des similarités et des différences

L'évaluation de la similarité des motifs (et de leurs différences) est réalisée dans la troisième étape de notre méthode d'appariement en confrontant les motifs par paires. Nous rappelons que la comparaison de paires d'éléments ou de structures, qui a pour but de leur affecter un degré « d'alignabilité » (ou score de similarité), peut être faite de nombreuses façons et en se basant sur des aspects variés [Djourak Kengue *et al.* 2008]. Les critères des approches classiques d'appariement de modèles sont relatifs de manière générale aux étiquettes, aux types, aux relations sémantiques entre les concepts ou à leur position dans les modèles.

La définition d'une mesure de similarité sur les éléments des modèles est une expression combinant (par exemple, dans une somme ou une combinaison linéaire) les similarités de chacune des dimensions descriptives choisies pour l'appariement. Par exemple, pour les classes, il est possible de se focaliser uniquement sur les noms et les attributs, en ignorant les extrémités des associations et les classes mères éventuelles. La comparaison sur chacune des dimensions requiert une fonction dédiée. Dans notre cas, la mesure de similarité de deux motifs est réalisée par le biais d'une

fonction de similarité paramétrable calculant un score de similarité, positif et limité par un seuil maximum, propre à l'ensemble de leurs dimensions descriptives dont les structures internes et externes du couple de motifs comparés.

### 5.4.1 Problème de dépendance mutuelle dans l'évaluation des similarités

Même si les approches d'appariement ne le précisent pas explicitement, toutes les dimensions descriptives participent *a priori* à l'expression de la similarité de deux éléments. Bisson énonce notamment que deux éléments au sein d'une représentation structurée sont d'autant plus similaires que les éléments qui l'entourent le sont aussi [Bisson 1992]. Djourak Kengue *et al.* parlent de renforcement mutuel lorsque la mesure de similarité implante le principe suivant : deux éléments sont d'autant plus similaires que les éléments directement reliés à ceux-ci le sont aussi [Djourak Kengue *et al.* 2008]. Une telle définition mène à une dépendance mutuelle et récursive entre les scores de similarité. Par exemple, les scores de similarité de deux classes et de leurs attributs dépendent les uns des autres. À cause de la circularité, le calcul de scores ne pourra pas toujours être effectué de façon directe et exacte. La circularité peut être évitée soit en imposant des structures descriptives qui ne comportent pas de cycle[19], soit en utilisant deux fonctions différentes pour les éléments, une en tant que couple comparé et une autre en tant que contributeurs à la similarité d'un couple voisin [Djourak Kengue *et al.* 2008].

Un autre problème renforce l'idée de prendre en compte le voisinage des éléments en complément de leurs étiquettes (les noms sont très importants pour discriminer deux éléments) lors de la mesure de leurs similarités et de leurs différences : des concepts peuvent être implicites dans l'énoncé à modéliser, l'objectif étant pour l'apprenant de les découvrir et de les modéliser correctement. Dans ce cas de figure, les noms employés par l'apprenant s'il identifie ce concept peuvent n'avoir aucun rapport avec ceux employés dans une ou plusieurs références. Dans l'exercice « Stylo et Feutre » (cf. partie 1.4), les classes « Stylo » et « Feutre » ont des attributs (« couleur » et « marque ») et un comportement commun (des liens sémantiques avec les concepts de « Personne » et de « Corps »). Ces propriétés et ce comportement ne doivent pas être représentés en doublon dans le diagramme de classes mais sous la forme d'une classe les factorisant. Dans la référence proposée, cette classe est nommée « Crayon » mais elle pourrait être nommée autrement (par exemple « Outil dactylographique », « Outil d'écriture »...). La seule comparaison des noms ne permettra pas de relever l'appariement de ces classes. Dans le cas où cette classe n'est pas modélisée dans le diagramme de l'apprenant, son comportement et ses propriétés en découlant ont pu être déplacés et représentés en doublon dans différentes classes devant en hériter (« Stylo », « Feutre », « Feutre effaceur »). Ces différences imposent de prendre en compte le voisinage des éléments pour pouvoir être relevées par un processus d'appariement.

### 5.4.2 Principe général du calcul de scores de similarité

L'organisation des éléments dans les diagrammes de classes fait que le problème de circularité ne peut être traité au niveau de la représentation des modèles : en effet, les diagrammes de classes présentent des cycles, des boucles et les relations d'association ne forment pas des hiérarchies arborescentes. Nous avons donc choisi de séparer dans notre approche les critères qui ne dépendent pas du voisinage des motifs de ceux qui en dépendent. Nous considérons deux scores calculables lors de la mesure de similarité de chaque paire de motifs, d'une part un score simple et d'autre part

[19] Dans de très nombreuses approches d'appariement, les modèles à apparier sont représentés sous forme de DAG (*Directed Acyclic Graph*). Le problème de circularité est ainsi très limité par la représentation retenue pour les modèles.

un score complexe. Les scores simples et complexes sont normalisés par rapport au nombre de critères utilisés lors de l'évaluation de leurs similarités et de leurs différences. La combinaison des deux scores constitue le score de similarité total du couple de motifs *(s,c)* comparé. La fonction de calcul à un niveau abstrait peut être exprimée de la sorte :

$$score\ de\ similarité(s,c) = (score\ simple\ (s,c) + score\ complexe\ (s,c)) / nb.\ critères\ total$$
$$nb.\ critères\ total = nb.critères\ du\ score\ simple + nb.\ critères\ du\ score\ complexe$$

Le **score simple** exprime la pondération des dimensions descriptives du couple de motifs *(s,c)* indépendantes de tout autre motif des modèles. Ces critères sont essentiellement quantitatifs : pour les motifs simples, ils sont relatifs au nombre de relations liées, au nombre d'éléments encapsulés, à la visibilité, à l'abstraction, au sens d'orientation, aux types, etc. L'évaluation des similarités des espaces de nommage est réalisée lors du calcul du score simple car les noms employés ne dépendent pas du reste des diagrammes. Chaque valeur calculée par rapport à une dimension (un critère) est pondérée. Plus un critère a un poids élevé plus il aura d'impact sur le score total. Les valeurs des critères pris en compte dans le score simple constituent une partie calculable immédiatement de la valeur finale.

$$score\ simple(s,c) = poids\ critère_1 \times simCritère_1(s,c) + ... + poids\ critère_n \times simCritère_n(s,c)$$

Le **score complexe** agrège une partie des scores des motifs en contexte avec le couple de motifs comparés *(s,c)*. Sans prendre en compte la nature des motifs comparés, le contexte est constitué de leurs motifs contenus, conteneurs et liés. L'évaluation des similarités des motifs appartenant au contexte est contrainte par le type des motifs à prendre en compte. Par exemple, les relations liées à des classes sont comparées et appariées entre elles et les attributs d'une classe d'un diagramme sont comparés et appariés avec les attributs d'une classe de l'autre diagramme, etc. Sous forme abstraite, le calcul du score complexe du couple *(s,c)* est réalisé ainsi :

$$score\ complexe(s,c) = \sum_{type} simContenus(s,c,type) + \sum_{type} simConteneurs(s,c,type) + \sum_{type} simLiens(s,c,type)$$

La valeur de similarité retournée par chacune des trois fonctions *simContenus*, *simConteneurs*, *simLiens* est normalisée en interne par le nombre de motifs locaux appariés par paires. En fonction du paramétrage du système d'appariement, le score peut être déprécié par le nombre de motifs non appariés. Seul un pourcentage des valeurs de similarité participe au calcul du score complexe. Ces différentes variables sont paramétrables avant le lancement de la mesure de similarité et restent ensuite fixes tout au long du processus.

### 5.4.3 Critères pris en compte pour évaluer les similarités et les différences

La fonction de similarité utilisée pour chaque couple de motifs est contextualisée par rapport aux critères qui leur sont propres. Ces critères sont extraits directement de la sémantique du métamodèle des modèles à comparer pour les motifs simples. Les principaux critères par type d'éléments UML sont les suivants :

- Une classe est définie par un nom, une abstraction, une visibilité, un ensemble d'attributs et d'opérations qu'elle encapsule et par les relations qui la lient aux autres classes du diagramme ;
- Une relation d'héritage est définie par un sens d'orientation et par ses classes liées ;
- Une relation de dépendance est définie par un nom (ou un stéréotype), une orientation et par ses classes liées ;

- Une relation d'association peut être binaire (liée à deux classificateurs) ou n-aire (liée à plus de deux classificateurs). Elle est définie par un nom, un type d'agrégation (composition, agrégation ou aucun type d'agrégation), des fins d'association (multiplicités et rôles) et les classes qu'elle relie ;
- Une classe d'association est définie par un nom, une abstraction, une visibilité, un ensemble d'attributs et d'opérations, des fins d'association (multiplicités et rôles) qu'elle encapsule et par les classes qu'elle lie ;
- Une caractéristique UML est définie par un nom, une visibilité, et par le classificateur qui la contient. Un attribut est défini en plus par type et une opération par des paramètres.
- Une fin d'association est définie par un rôle (un nom), une multiplicité, un type d'agrégation, l'association qui la contient et la classe qu'elle lie à l'association.

La similarité de deux motifs complexes est évaluée en fonction des motifs simples qu'ils contiennent, ceux qui les lient à d'autres motifs complexes (ces motifs simples jouant le rôle de pont entre deux motifs complexes). Si les motifs complexes admettent des relations de contenance les uns par rapport aux autres (les hiérarchies d'héritage multiple encapsulent plusieurs arbres d'héritages) alors ceux qu'ils contiennent et les motifs complexes les contenant (les motifs complexes parents) peuvent être pris en compte lors de l'évaluation de leurs similarités et de leurs différences.

### *5.4.4 Evaluations des similarités et des différences à l'aide de comparateurs*

Nous pouvons remarquer que certains des critères que nous venons de présenter dans l'évaluation des similarités et des différences ne sont pas communs à tous les éléments. Au contraire d'autres critères sont communs à plusieurs types d'éléments. De plus, il est nécessaire dans notre cas de pouvoir évaluer la similarité des éléments de nature différente étant donné qu'un concept peut par exemple être représenté par un classificateur dans un diagramme et sous la forme d'un attribut dans l'autre diagramme. Dans notre approche, l'évaluation des similarités de deux motifs est exprimée par la somme des résultats de plusieurs méthodes de calcul. Chaque méthode se focalise sur une ou plusieurs dimensions des motifs à apparier.

Nous proposons des comparateurs pour prendre en charge l'évaluation des similarités et des différences locales aux modèles, aux motifs, et aux chaînes de caractères à apparier. Nous avons défini la hiérarchie de comparateurs suivante pour mener la comparaison des constituants des diagrammes de classes UML.

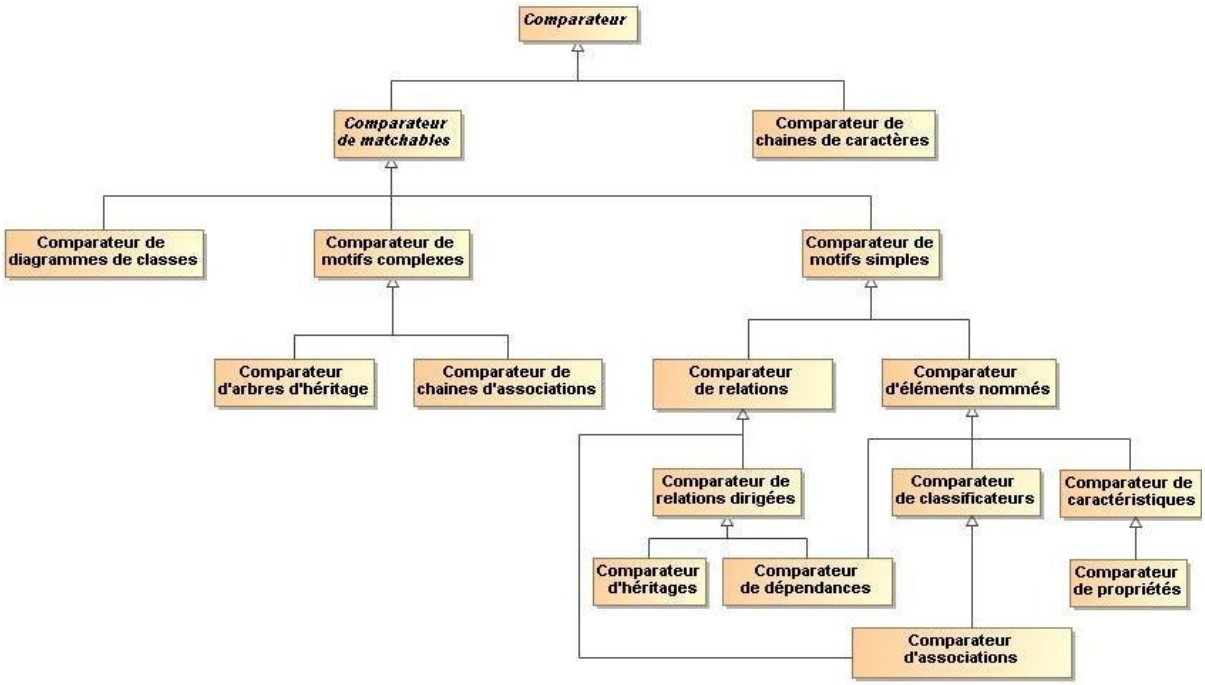



**Figure 41 : Hiérarchie des comparateurs dédiés à la mesure de similarité dans ACDC**

Tous les comparateurs ont une structure et un fonctionnement général commun pour évaluer les similarités et les différences des objets à comparer. Un comparateur est instancié pour chaque ensemble de motifs à comparer. Chaque comparateur mémorise les objets qu'il compare, les scores de similarité calculés et les différences locales identifiées pour éviter de les recalculer lors des comparaisons d'autres objets en dépendant. Si plusieurs objets ont déjà été comparés et qu'ils participent à la confrontation d'autres objets alors le comparateur existant de ces objets est consulté. Au besoin, les scores et les différences locales calculés sont mis à jour lorsque des problèmes de circularité se présentent et que le calcul est partiel.

Le comportement d'évaluation des similarités et d'identification des différences locales aux objets comparés est défini dans le *Comparateur de matchables*[20] sous forme de trois méthodes distinctes nommées respectivement *compare*, *compareSimpleScore* et *compareComplexScore*. La fonction *compare* définit la manière dont les résultats des fonctions *compareSimpleScore* et *compareComplexScore* sont agrégés et dans quel ordre les calculs sont effectués. Les fonctions *compareSimpleScore* et *compareComplexScore* sont définies ou redéfinies dans chaque comparateur individuel en fonction des dimensions qu'ils prennent en compte. Les calculs sont complétés au fur et à mesure en invoquant les méthodes de calcul des comparateurs parents. Nous détaillons dans l'annexe 5 les critères utilisés par chaque comparateur et la manière dont les comparateurs opèrent pour évaluer les similarités et les différences des objets qu'ils ont à comparer.

Tous les comparateurs ont été définis pour fonctionner de manière univoque (comparaison de deux objets) ou de manière multivoque (comparaison de m objets avec n objets). Si deux objets sont comparés (cardinalité 1:1) alors le score simple est calculé (si aucune demande de comparaison n'a été faite auparavant sur ces objets) en appelant la méthode *compareSimpleScore*. Ensuite le score complexe est évalué par la méthode *compareComplexScore*. Si

[20] Les *matchables* (tiré de l'anglais *match*) sont des objets appariables dans notre méthode d'appariement, c'est-à-dire à la fois les modèles (diagrammes de classes UML dans notre cas), les motifs complexes et les motifs simples.

plusieurs objets doivent être comparés en même temps (cardinalités 1:n, n:1 et m:n) alors dans un premier temps pour chaque couple possible d'objets, un comparateur est invoqué (ou consulté s'il est préexistant) et est en charge d'évaluer leurs similarités et leurs différences. Lorsque tous les objets ont été comparés de cette manière, une matrice des scores de similarité de m lignes et n colonnes est constituée. Trois stratégies[21] ont été étudiées pour combiner les résultats :

- **Sélection du meilleur appariement univoque :** cette première stratégie ne retient que le couple d'objets ayant la meilleure similarité (i.e. le score de similarité ayant la valeur la plus proche de 0). Elle permet de sélectionner uniquement le couple d'objets maximisant la similarité parmi plusieurs ensembles de motifs.
- **Sélection du meilleur ensemble d'appariements univoques :** chaque objet source est associé au plus à un objet cible et réciproquement en ne retenant que les couples ayant les meilleurs valeurs de similarité. La valeur de similarité de l'ensemble est dans ce cas est une moyenne des couples les plus similaires. Cette stratégie est utilisée pour qualifier les appariements univoques locaux aux motifs. Par exemple, si une classe du $DC_S$ est comparée à une classe du $DC_C$, alors le calcul du score complexe de ces deux classes prend en compte uniquement les alignements univoques des relations et caractéristiques maximisant les similarités des classes.
- **Agrégation des résultats :** les similarités de tous les couples sont prises en compte et cumulées. Le score obtenu est une valeur moyenne de tous les couples possibles. L'hypothèse ici est que tous les appariements possibles participent à la similarité de l'ensemble. Cette contrainte est retenue notamment pour déterminer et qualifier des appariements multiples de cardinalité 1:n et n:1 dans la phase d'appariement final (cf. partie 5.5).

Le comportement de notre mesure de similarité est distribué dans l'ensemble des comparateurs (ce qui la rend difficile à appréhender). Mais cette technique présente l'avantage de pouvoir évaluer les similarités et les différences d'éléments de natures différentes en se focalisant uniquement sur les critères qu'ils ont en communs. Par exemple, la comparaison d'une relation d'héritage avec une relation d'association est réalisée automatiquement par le biais du comparateur de relations. La fonction utilisée pour les comparer pondère et agrège uniquement les critères communs à toutes les relations, c'est-à-dire le fait qu'elles soient liées à un certain nombre de classificateurs et que par conséquent une partie de leur sens découle de leurs classificateurs liés. Toutefois, deux relations d'association sont évaluées par le comparateur d'associations qui leur est spécifique. Ce dernier utilise en cascade les résultats des comparateurs d'éléments, d'éléments nommés, de classificateurs et de relations qu'il agrège aux autres critères qui dépendent spécifiquement des associations : leur type d'agrégation et leurs multiplicités.

### 5.4.5 Comparaison spécifique des espaces de nommage

Les éléments d'un diagramme de classes UML (classes, attributs, opérations et relations) sont généralement nommés. Les noms qualifiant les éléments ne sont pas de simples étiquettes appartenant à un ensemble fini comme c'est le cas dans de nombreux graphes étiquetés. En effet dans les diagrammes UML, les espaces de nommage se réfèrent à des objets du monde ou à des concepts. De nombreuses possibilités (cf. les approches linguistiques dans la partie 4.2.2.1) existent pour apparier les espaces de nommage. La plus simple est la comparaison de ceux-ci en tant que

[21] Ces stratégies ont été définies pour prendre en compte le contexte des motifs comparés (i.e. les motifs contenus et appartenant à leur voisinage). Il ressort de nos tests que la valorisation d'un seul couple de motifs entrave le plus souvent l'identification des différences de structure des diagrammes. Cette stratégie a donc été substituée par les deux dernières plus tolérantes aux écarts de représentation. La deuxième stratégie est utilisée dans la mesure de similarité pour déterminer les appariements locaux aux couples de motifs comparés (cf. partie 5.4.6). La stratégie d'agrégation est en revanche trop « globale » pour être pertinente lors de cette même tâche (trop de scores non pertinents seraient pris en compte). En revanche, elle se revèle beaucoup plus intéressante lors de la détermination des appariements multivoques du *mapping* étant donné qu'une partie des appariements de motifs ont déjà été fixés.

chaînes de caractères. Cependant, une telle comparaison risque d'être parfois trompeuse. Une approche alternative consiste à utiliser un thesaurus ou autre ressource lexicale (cf. WordNet dans la partie 3.3.1) afin de capter les rapports entre les termes (cf. synonymie, homonymie, hyponymie, hyperonymie, etc.). Bien que plus puissante, cette approche se heurte aussi à l'ambiguïté intrinsèque de la langue, d'où l'impossibilité de limiter la confrontation aux simples noms [Djourak Kengue *et al.* 2008].

Dans notre contexte, les énoncés des exercices que l'apprenant modélise sont définis librement par l'enseignant et ne sont pas dépendants d'un domaine donné. Nous ne disposons pas et n'utilisons donc pas de thesaurus en langue française qui seraient spécifiques aux domaines de connaissance des exercices que les apprenants ont à modéliser. Lors de la construction de son diagramme, l'étudiant peut utiliser des synonymes à la place des noms employés par l'enseignant dans le diagramme de référence. De plus, les étudiants font des fautes de frappe et d'orthographe. Il est donc nécessaire de mettre en place une comparaison spécifique et robuste des noms des éléments pour pouvoir les apparier correctement.

Dans Diagram (cf. chapitre 2.4.3), lors de l'élaboration d'un diagramme de référence à partir de l'énoncé, l'enseignant a la possibilité d'associer des expressions de l'énoncé aux éléments du diagramme. Ces informations peuvent être utiles lors de la comparaison du diagramme de l'apprenant conçu dans les mêmes conditions notamment si l'apprenant a associé des éléments de son diagramme avec des expressions similaires à celles employées dans la référence. Nous utilisons dans l'ACDC ces expressions comme une base simpliste d'espaces de nommage alternatifs lors de l'appariement des noms des éléments.

Un appariteur spécifique aux chaînes de caractères similaire à ceux définis dans [Do & Rahm 2007] [Giunchiglia *et al.* 2007] a été conçu et inclus dans notre approche. Il est capable de rechercher les sous-chaînes communes des espaces de nommage. Il calcule ensuite un score pour chaque couple d'espaces de nommage (pris en compte dans la mesure de similarité des éléments nommés) et sélectionne le couple dont le score de similarité est le meilleur (le plus faible dans notre méthode). Au préalable, les espaces de nommage sont filtrés par casse, genre et nombre pour ne pas entraver leur comparaison. Les caractères spéciaux et les séparateurs y sont également supprimés pour les mêmes raisons.

Par exemple, une classe nommée « Chauffeur de poids lourd » dans le diagramme de référence peut avoir été nommée par l'apprenant « chauffeurs de poid lourd » ou « chaufeur de camions ». Les noms de ces classes sont filtrés et deviennent respectivement « chauffeurdepoidlourd », « chauffeurdepoidlourd » et « chaufeurdecamions ». Les deux premières chaînes de caractères après filtrage sont identiques et ne demandent donc pas plus de traitement pour être appariées. Dans le cas des chaînes « chauffeurdepoidlourd » et « chaufeurdecamions », il est nécessaire d'identifier leurs sous-chaînes communes pour évaluer leurs similarités. Les sous-chaînes communes sont ici « chauf » et « eurde » et correspondent à la moitié des caractères constituant ces deux noms. La similarité n'est donc pas optimale mais n'est pas négligeable si aucun autre élément de la référence n'a une meilleure similarité avec « chaufeurdecamions ».

L'algorithme général de notre mesure de similarité de deux chaînes de caractères incluant le filtrage et la recherche des sous chaînes communes est reporté dans l'annexe 6. Pour l'extraction des sous-chaînes communes, il s'inspire du MCWPA (***M**oving **C**ontracting **W**indow **P**attern **A**lgorithm*) [Yang *et al.* 2002]. Nous avons expérimenté

plusieurs formules de calcul pour évaluer la similarité de deux chaînes de caractères dont le calcul du SSNC (***S**um of the **S**quare of the **N**umber of the same **C**haracters*) et une formule en découlant contrainte par un seuil.

### 5.4.6 Appariements locaux des motifs contenus, conteneurs et liés

Nous avons expliqué précédemment que dans notre mesure de similarité le score complexe des motifs prend en compte les motifs contenus, conteneurs et liés appariés par paires en respectant une contrainte de type. L'organisation des motifs simples sous forme de motifs complexes prend toute son importance dans ce calcul. En effet, non seulement les motifs du contexte directement liés aux deux motifs sont comparés entre eux mais ils pourront en fonction du paramétrage du système être confrontés à un contexte plus large par propagation dans les hiérarchies d'héritages et par relaxation dans les chaînes d'associations (cf. partie 5.2.4). Par propagation descendante (i.e. vers les enfants), ascendante (i.e. vers les parents) et transversale (i.e. vers les frères)[22], le contexte du motif source est comparé également aux motifs du contexte des parents, des fils, des frères du motif cible et réciproquement. Le contexte propagé ou dérivé d'un motif simple est récupéré en interrogeant les motifs complexes qui le contiennent. Ce procédé permet de prendre en compte la structure externe des motifs les uns par rapport aux autres et ainsi d'identifier des déplacements d'éléments dans les diagrammes. Les comparateurs de classificateurs qualifient les différences au niveau des relations qui devraient être liées et des caractéristiques qui devraient être contenues. Les comparateurs de relations qualifient des différences par rapport aux classificateurs qui devraient leur être liés. Enfin les comparateurs spécifiques aux caractéristiques qualifient des différences relatives à leurs conteneurs (les différences sont exposées dans la partie 5.6).

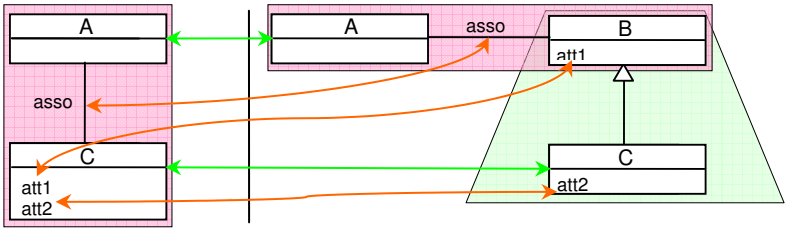



**Figure 42 : Exemple de diagramme de classes simple altéré structurellement**

Nous illustrons l'influence du contexte sur la détermination des appariements locaux des diagrammes en confrontant les deux diagrammes de la figure 42. Nous notons CA l'unique chaîne d'associations apparaissant dans les deux diagrammes et AH l'arbre d'héritages du diagramme de droite (le diagramme de référence). Nous avons volontairement reporté les mêmes noms fictifs pour les classes, les attributs, et les relations d'association de ces deux diagrammes pour ne prendre en compte ici que les aspects structurels et ne laisser aucune ambiguïté au niveau de l'appariement de leurs étiquettes (i.e. les noms des éléments). Chaque ligne illustre la mesure des similarités et des différences des motifs contenus, conteneurs ou liés d'un couple de motifs simples (première colonne).

De manière générale, la prise en compte de la propagation du contexte permet de qualifier de meilleurs appariements locaux lorsque les structures sont altérées (cf. la cinquième colonne sans contexte et la sixième avec contexte). Les omissions et les insertions en rouge dans le tableau signifient que les motifs peuvent être appariés entre

[22] La propagation des propriétés et du comportement d'une classe se fait naturellement de manière descendante dans les hiérarchies d'héritages. Nous mettons en place également en place un mécanisme de propagation ascendante et transversale car les attributs, les opérations et les relations peuvent avoir été représentés par l'apprenant à des « endroits » différents dans les hiérarchies de classes. L'identification de ces déplacements est facilitée grâce à ce procédé.

eux si le poids affecté à la similarité des noms des éléments participe faiblement à la similarité totale (i.e. les aspects structurels prédominent dans le choix des appariements locaux dans ce cas).

| Motifs simples (s,t) | Motifs contenus / conteneurs / liés de (s,t) | Motifs propagés des parents de (s,t) | Motifs propagés des enfants de (s,t) | Appariements locaux et différences locales sans propagation du contexte | Appariements locaux et différences locales avec propagation du contexte |
|---|---|---|---|---|---|
| (A,A) | ((asso),(asso)) | | | (asso,asso) | (asso,asso) |
| (A,B) | ((asso),(asso,h))<br>(-,(att1)) | | (-,(att2)) | (asso,asso)<br>(h) : Omission<br>(att1) : Omission | (asso,asso)<br>(h) : Omission<br>(att1) : Omission |
| (A,C) | ((asso),(h))<br>(-,(att2)) | (-,(asso))<br>(-,(att1)) | | (asso) : Insertion<br>(h) : Omission<br>(att2) : Omission | (asso,asso) : Transfert vers fils<br>(h) : Omission<br>(att2) : Omission |
| (C,A) | ((asso),(asso))<br>((att1,att2),-) | | | (asso,asso)<br>(att1) (att2) : Insertion | (asso,asso)<br>(att1) (att2) : Insertion |
| (C,B) | ((asso),(asso,h))<br>((att1,att2),att1) | | (-,(att2)) | (asso,asso)<br>(h) : Omission<br>(att1,att1)<br>(att2) : Insertion | (asso,asso)<br>(h) : Omission<br>(att1,att1)<br>(att2,att2) : Transfert vers père |
| (C,C) | ((asso),(h))<br>((att1,att2),(att2)) | (-,(asso))<br>(-,(att1)) | | (asso) : Insertion<br>(h) : Omission<br>(att1) : Insertion<br>(att2,att2) | (asso,asso) : Transfert vers fils<br>(h) : Omission<br>(att1,att1) : Transfert vers fils<br>(att2,att2) |
| (asso,h) | ((A,C),(B,C)) | | | (A) : Insertion<br>(B) : Omission<br>(C,C) | (A) : Insertion<br>(B) : Omission<br>(C,C) |
| (asso,asso) | ((A,C),(A,B)) | | (-,(C)) | (A,A)<br>(C) : Insertion<br>(B) : Omission | (A,A)<br>(C,C) : Transfert vers père<br>(B) : Omission |
| (att1,att1) | ((C),(B)) | | (-,(C)) | (C) : Insertion<br>(B) : Omission | (C,C) : Transfert vers père<br>(B) : Omission |
| (att2,att1) | ((C),(B)) | | (-,(C)) | (C) : Insertion<br>(B) : Omission | (C,C) : Transfert vers père<br>(B) : Omission |
| (att2,att2) | ((C),(C)) | (-,(B)) | | (C,C) | (C,C) |
| (att1,att2) | ((C),(C)) | (-,(B)) | | (C,C) | (C,C) |

**Figure 43 : Appariements locaux des motifs simples des diagrammes de la figure 42**

L'évaluation des similarités locales des motifs complexes est présentée de la même manière dans le tableau de la figure 44. Le contexte des motifs complexes concerne les relations externes aux motifs et les classes liées à ces relations en dehors du motif complexe. Comme pour les motifs simples, la prise en compte du contexte améliore les appariements locaux relevés et les différences les qualifiant.

| Motifs complexes (s,t) | Motifs simples contenus | Motifs complexes liés directement | Motifs simples liés directement | Appariements locaux et différences locales sans contexte | Appariements locaux et différences locales avec contexte |
|---|---|---|---|---|---|
| (CA,CA) | ((A,C),(A,B))<br>((asso),(asso)) | (-,(AH)) | (-,(C))<br>(-,(h)) | (A,A)<br>(C) : Insertion<br>(B) : Omission<br>(asso) : Insertion<br>(asso) : Omission | (A,A)<br>(C,C) : Remplacement<br>(B) : Omission<br>(asso) : Insertion<br>(asso) : Omission |
| (CA,AH) | ((A,C),(B,C))<br>((asso),(h)) | (-,(CA)) | (-,(A))<br>(-,(asso)) | (A) : Insertion<br>(C,C)<br>(B) : Omission<br>(asso) : Insertion<br>(h) : Omission | (A,A) : Remplacement<br>(C,C)<br>(B) : Omission<br>(asso,asso) : Transfert vers fils<br>(h) : Omission |

**Figure 44 : Appariements locaux des motifs complexes des diagrammes de la figure 42**

## 5.5 Choix de l'appariement des motifs

La phase d'évaluation des similarités et des différences permet d'obtenir un pré-appariement univoque des motifs à l'échelle du couple de diagrammes de classes ($DC_S$, $DC_C$) grâce au comparateur de diagrammes et une vision du problème d'appariement à différentes échelles des diagrammes comparés, c'est-à-dire au niveau des motifs complexes et des motifs simples : les classificateurs, les relations et les caractéristiques. Ces vues déterminées de manière indépendante présentent le plus souvent des conflits étant donné que chaque couple de motifs confrontés a une vision locale de l'appariement de ses motifs contenus et des motifs appartenant à son voisinage (parents, enfants, frères, motifs liés directement ou indirectement). Dans ce contexte, la phase de choix de l'appariement final a pour objectif de retenir un *mapping*[23] cohérent des motifs précédemment comparés sous la forme d'une liste d'appariements qualifiés par des différences. Les appariements et les différences du *mapping* sont univoques pour les plus simples et multivoques pour les plus complexes.

La première difficulté de cette tâche réside dans le filtrage et la sélection des couples de motifs (appariements univoques) en veillant à ce que le *mapping* constitué soit cohérent, qu'il préserve au maximum les similarités et que leurs différences soient correctement qualifiées. La seconde difficulté est de déterminer des appariements multivoques pertinents améliorant le *mapping* en veillant à maintenir sa cohérence globale et locale. Nous avons choisi de mener la construction du *mapping* tout d'abord pour les motifs complexes puis pour les motifs en suivant quatre étapes générales séquentielles sans *backtracking* (i.e. sans retour en arrière) :

- Définition d'une liste de couples de motifs candidats qualifiés par des différences générales ;
- Sélection des appariements univoques des motifs ;
- Détermination des appariements multivoques des motifs ;
- Finalisation du *mapping*.

Dans notre proposition, l'appariement est mené des structures les plus générales vers les structures les plus spécifiques (*top-down*). Le *mapping* est nourri au fur et à mesure des étapes par deux appariteurs distincts. Un premier appariteur nommé « Appariteur de motifs complexes » est lancé pour déterminer les appariements univoques et multivoques des motifs complexes. Le second appariteur nommé « Appariteur de motifs simples » est exécuté ensuite pour faire de même sur les motifs simples. L'appariteur de motifs simples repose sur les appariements préalablement identifiés au niveau des motifs complexes pour identifier et qualifier les appariements des motifs simples les constituant.

Dans la suite de cette partie, nous décrivons tout d'abord la manière dont les listes de couples de motifs candidats sont constituées. Ensuite, nous expliquons le fonctionnement des appariteurs et nous détaillons la manière dont les appariements univoques sont sélectionnés et ensuite « évolués » en appariements multivoques. Enfin nous décrivons le cas particulier des appariements hétérogènes (appariement de motifs simples de types différents).

[23] Nous parlons de *mapping* car les différences qualifiant les appariements sont exprimées par rapport au diagramme $DC_S$ (par exemple, « Insertion » ou « Omission » d'un motif dans le $DC_S$).

### 5.5.1 Définition des listes de couples de motifs candidats à l'appariement

Le processus de définition de la liste de couples de motifs candidats à l'appariement est général aux deux types d'appariеurs. La liste est constituée à partir du pré-appariement du couple ($DC_S$, $DC_C$), d'un *mapping* partiel (il est vide dans le cas de la définition des motifs complexes à l'appariement) et des vues locales des couples de motifs comparés. Pour cela, les appariements locaux qualifiés par des différences sont récupérés sans filtrer les redondances d'une part du $DC_S$ vers le $DC_C$ et d'autre part du $DC_C$ vers $DC_S$ (pour récupérer tous les appariements possibles). Ces deux ensembles de couples sont ensuite fusionnés pour constituer la liste de couples de motifs candidats à l'appariement du couple ($DC_S$, $DC_C$). Des redondances sont présentes dans les ensembles et les couples ne sont pas classés dans un ordre précis. Lors de la fusion des deux listes, les couples identiques (redondants) sont comptées et les doublons sont supprimés. Le nombre de redondances est mémorisé pour chaque couple.

Nous faisons l'hypothèse que plus un couple est présent dans les différentes échelles des appariements locaux (plus un couple est redondant en conséquence dans les deux listes à fusionner), plus il aura de chance d'être pertinent dans le *mapping* final. Les couples faiblement représentés sont mineurs car ils n'interviennent que dans le contexte d'un très faible nombre de couples de motifs comparés et sont vraisemblablement des incohérences. Les couples les plus représentés doivent donc être évalués en premier.

Pour être parcourue par les appariеurs, la liste de couples est triée tout d'abord par type de motifs. L'ordonnancement des types de motifs est le suivant : pour les motifs complexes, les hiérarchies d'héritages sont reportées en premier puis les chaînes d'associations. Les motifs simples sont classés dans cet ordre : classificateurs, relations et caractéristiques. Puis pour chaque type de motifs, les motifs sources apparaissant dans le plus de couples sont classés en premier. Enfin pour chaque motif source (*s*), les couples avec les motifs cibles (*c*) sont classés par type de différences qualifiant la correspondance (les différences préservant le plus la structure sont classées en premier). Si plusieurs couples entre un motif *s* et différents motifs *c* sont qualifiés par une différence générale « Void » (cf. partie 5.6.2) alors ils sont classés par nombre décroissant de redondances (i.e. le premier est le plus redondant). Ensuite viendront les « Remplacements » et enfin les « Transferts ».

### 5.5.2 Comportement général des appariеurs

Le comportement général des appariеurs de motifs est glouton lors de la sélection des appariements univoques (couples) et lors de l'évaluation des appariements multivoques. Les appariеurs construisent le *mapping* au fur et à mesure du parcours de la liste des couples de motifs candidats qualifiés par une différence générale, en ajoutant les appariements univoques retenus. Les couples de motifs candidats sont ajoutés au fur et à mesure dans le *mapping final* sans avoir besoin de les rechercher. Pour cela, les appariеurs évaluent chaque couple à partir des scores de similarité[24] et vérifient la cohérence du couple par rapport au contenu du *mapping* courant. Les appariements retenus ont donc un « bon » score de similarité et sont cohérents avec les appariements et les différences (les qualifiant) préalablement identifiés. Une fois que tous les appariements univoques ont été sélectionnés, les appariements multivoques sont

[24] La prise en compte du score peut être réalisée sur le score complet (i.e. les scores simples et complexes) ou ne reposer que sur une partie du score (i.e. le score simple ou le score complexe) pour orienter plus ou moins l'appariement sur les structures internes ou externes des motifs. Nous avons également introduit la possibilité de prendre en compte à part la similarité des noms pour éliminer ou sélectionner des candidats à l'appariement dans le *mapping*.

déterminés à partir des motifs non appariés jusque là en les confrontant aux appariements univoques du *mapping* courant. Les appariements univoques servant de base aux appariements multivoques retenues sont retirés du *mapping* courant pour être remplacés par les appariements multivoques qualifiés par des différences multivoques. Pour finir, la finalisation du *mapping* est effectuée : si des motifs n'ont pas été appariés de manière univoque ou multivoque, ils sont ajoutés au *mapping* final sans équivalent dans l'autre diagramme et sont qualifiés avec des différences univoques unitaires d'insertion ou d'omission. L'apparieur de motifs complexes suit exactement les quatre étapes que nous venons de décrire dans ce paragraphe.

### 5.5.3 Comportement spécifique de l'apparieur de motifs simples

L'apparieur de motifs simples ordonne plusieurs phases séquentielles d'appariement univoque et multivoque sur les différents types de motifs simples existant dans le diagramme de classes pour nourrir le *mapping*. Il suit les étapes suivantes :

- Définition de la liste d'appariements candidats pour les motifs simples ;
- Création de quatre tables de correspondance des noms des motifs simples ;
- Sélection des appariements univoques de classes ;
- Détermination des appariements multivoques de classes ;
- Sélection des appariements univoques de relations ;
- Sélection des appariements univoques de caractéristiques ;
- Détermination des appariements univoques hétérogènes ;
- Détermination des appariements multivoques de relations ;
- Détermination des appariements multivoques de caractéristiques ;
- Finalisation du *mapping*.

Avant de débuter le processus d'appariement glouton, l'apparieur de motifs simples crée quatre tables de correspondance possibles des noms employés et des expressions textuelles ayant permis de les créer entre les deux diagrammes selon une direction de mise en correspondance (du $DC_S$ vers le $DC_C$ ou du $DC_C$ vers le $DC_S$) et selon deux autres critères : la **présence de sous chaînes communes d'une taille minimum** et le **pourcentage de caractères communs minimum**. Selon le critère de sous-chaînes communes, toutes les chaînes de caractères en relation avec un des diagrammes admettant au moins une sous-chaîne commune de longueur $l$ (par exemple 4) avec une chaîne de caractères de l'autre diagramme sont retenues. Selon le second critère, toutes les chaînes de caractères ayant un pourcentage $p$ (par exemple 80%) en commun avec une autre chaîne de caractères sont retenues pour cette dernière. Chaque ligne de ces tables a pour clé une chaîne de caractères correspondant à un nom d'un élément d'un diagramme ou une expression textuelle de l'énoncé associé à un élément de ce même diagramme et comme valeur une liste de chaînes de caractères de l'autre diagramme vérifiant un des deux critères. Ces quatre tables sont consultées pour rejeter ou confirmer des appariements univoques, guider la création des appariements hétérogènes et multivoques.

L'apparieur de motifs simples prend en compte les classificateurs, les relations et enfin les caractéristiques de manière univoque puis de manière multivoque. Les motifs simples sont traités dans cet ordre car les classificateurs orientent les choix d'appariement des relations qui leur sont liées et des caractéristiques qu'ils encapsulent (les

caractéristiques peuvent être contenues également dans les relations d'association). Une étape intermédiaire est réalisée avant l'appariement multivoque des relations et des caractéristiques pour repérer les appariements dits hétérogènes, c'est-à-dire les motifs simples mis en correspondance ayant un type différent. Les motifs simples non appariés de manière univoques sont évalués sans contrainte de type dans cette phase (cf. partie 5.5.6).

### 5.5.4 Sélection des appariements univoques

La sélection des apparieurs univoques est menée en parcourant la liste triée des couples de motifs candidats. Pour chaque ensemble de couples possibles d'un motif source *s* du $DC_S$, une fonction *univalentMatch* est invoquée pour mener la sélection. Elle prend en paramètre le *mapping* courant et l'ensemble des couples possibles pour le motif *s*. Les couples sont parcourus de manière itérative jusqu'à ce qu'un couple soit retenu ou que tous les couples soient parcourus sans résultat.

Une couple *(s,c)* de la liste des candidats est retenue si deux conditions générales sont respectées (dans le cas contraire, le couple est rejeté) : la correspondance est cohérente avec le *mapping* courant et les motifs mis en correspondance ont une « bonne similarité ».

Pour savoir si une correspondance est cohérente, il faut vérifier tout d'abord si le motif cible *c* n'a pas déjà été sélectionné dans le *mapping* courant. Ensuite, les appariements locaux au couple *(s,c)* doivent être cohérents avec les appariements retenus dans le *mapping*. La vision à différentes échelles des pré-appariements possibles (définie lors de la phase de mesure des similarités et des différences locales) est utilisée pour maintenir la cohérence du *mapping* courant des diagrammes pendant son élaboration. En effet, lorsqu'un couple de motifs est ajouté à l'appariement courant, les appariements locaux du couple de motifs sont stockés dans une liste d'appariements « favorables » pour pouvoir orienter un peu plus l'appariement vers une solution cohérente. Par exemple, si deux classes nommées « feutre » et « Feutre » sont retenues dans l'appariement courant, les différences qualifiant les appariements locaux concernant leurs caractéristiques et relations communes (ou non) sont stockés pour être prises en compte lors de l'appariement des relations et des caractéristiques. A contrario, lorsqu'un couple est rejeté, ses appariements locaux et ses différences locales sont stockés dans une liste d'appariements « défavorables ». La vérification de la cohérence est menée par une fonction *isCoherent* prenant en paramètre la correspondance à évaluer et le *mapping* courant. Son stéréotype est le suivant :

*boolean isCoherent(correspondence, mapping)*

Une **bonne similarité** ne signifie pas nécessairement que le couple *(s,c)* sélectionné a le meilleur score de similarité par rapport à l'ensemble des couples possibles $(s_X,c)$ et $(s,c_Y)$. Ce concept est à mettre en corrélation tout d'abord avec la notion de **direction** du processus d'appariement. L'appariement peut être réalisé du motif *s* vers le motif *c* (s→c), du motif *c* vers le motif *s* (s←c) ou dans les deux sens (s↔c). En menant un appariement s→c, l'appariement veille à maximiser la similarité de *s* vis-à-vis de tous les motifs *c* (toutefois *c* pourrait s'apparier mieux à un autre motif $s_X$). Réciproquement en réalisant un appariement s←c, un apparieur veille à maximiser la similarité de *c* vis-à-vis de tous les motifs $s_X$ (cependant *s* pourrait s'apparier mieux à un autre motif $c_Y$). Enfin, le dernier cas, le plus contraint, limite le plus les erreurs d'appariement mais conduit à rejeter de nombreuses correspondances pouvant être

correctes car il maximise la similarité par rapport à *s* et *c* dans les deux directions (le score de similarité est le meilleur pour *s* et *c* vis-à-vis de tous les autres motifs $s_X$ et $c_Y$).

Quel que soit le choix retenu pour diriger l'appariement, un autre problème doit être géré : **la maximisation de la similarité**. Ce concept est relatif au fait d'autoriser qu'un ou plusieurs motifs puissent être sélectionnés comme candidats et ensuite ajoutés au *mapping*. Plusieurs stratégies sont applicables quelle que soit la direction d'appariement retenue. Elles ont été présentées et utilisées dans le système COMA [Do & Rahm 2002] [Do & Rahm 2007]. Ces trois stratégies respectivement ***MaxN***, ***MaxDelta*** et ***Threshold*** sont combinables en fonction des besoins d'appariement et des données sur lequel porte l'appariement :

- ***MaxN*** : les *N* motifs ayant les meilleurs scores de similarité par rapport à un autre motif de l'autre modèle sont des candidats possibles. *Max1* consiste à ne retenir que le couple ayant la meilleure similarité et garantit que les appariements retenus sont corrects mais ce choix est si contraint que de nombreuses correspondances ne peuvent pas être repérées. Au contraire, si le nombre *N* est trop grand, les appariements produits peuvent être de moins bonne qualité.
- ***MaxDelta*** : le motif avec la similarité maximale est retenu comme candidat possible ainsi que tous les autres motifs ayant une similarité dans une fourchette de tolérance *d* (une valeur absolue ou relative). La définition de la tolérance amène à contraindre plus ou moins la sélection des correspondances. Une tolérance trop large conduit à la sélection de nombreux candidats dont certains peuvent être faibles. Une tolérance très faible revient à utiliser *Max1*.
- ***Seuil (Threshold)*** : tous les motifs ayant un score de similarité ne dépassant pas un certain seuil *t* au niveau de leur score de similarité sont des candidats possibles. Cette stratégie prise en compte seule peut être trop laxiste si le seuil est trop élevé (production d'appariements erronés) et trop contraignante si le seuil est trop bas (un score bas signifie qu'il y a peu de différences entre les motifs comparés). Combinée à *MaxDelta* ou *MaxN*, un seuil adapté limite les choix d'appariements faibles d'un appariement.

Pour vérifier si la similarité de la correspondance courante est bonne vis-à-vis des autres correspondances existantes, nous gérons les trois directions d'appariement et les trois stratégies de sélection que nous venons d'exposer. Pour cela, nous avons défini une fonction nommée *haveGoodCurrentSimilarity* dont les paramètres d'entrée sont les motifs sources (*sources*), les motifs cibles (*targets*), la direction de l'appariement (*direction*) et la manière dont leur score est pris en compte, c'est-à-dire la prise en compte du score simple (*bWithSimpleScore*) et la prise en compte du score complexe (*bWithComplexScore*). Utiliser les deux scores revient à prendre en compte le score de similarité complet. Cette fonction a le stéréotype suivant :

*boolean haveGoodCurrentSimilarity(sources, targets, direction, bWithSimpleScore, bWithComplexScore)*

Nous avons choisi de retenir les appariements ayant une bonne similarité globale (prise en compte du score de similarité complet) dans l'un des deux sens (s→c ou s←c) ou dans les deux sens en prenant en compte uniquement le score complexe[25]. La fonction *haveGoodCurrentSimilarity* est appelée jusqu'à trois fois et évalue la pertinence de la correspondance du point de son score de similarité en utilisant les trois stratégies ***MaxN***, ***MaxDelta*** et ***Threshold***. Le

[25] Un appariement est retenu dans ce cas uniquement par rapport à son contexte. Il est nécessaire de contraindre plus fortement la direction pour éviter de sélectionner des appariements faibles. Ce critère est utile pour sélectionner des appariements d'éléments portant des noms différents mais représentant le même concept (par exemple des éléments non définis explicitement dans un énoncé ne sont pas toujours nommés à l'identique).

couple *(s,c)* est un bon candidat s'il fait partie des *N* couples admettant la meilleure similarité dans la direction choisie et si son score de similarité est compris dans une fourchette de tolérance relative au couple $(s_X,c)$ ou/et $(s,c_Y)$ ayant le meilleur score de similarité ou si le score est inférieur à un seuil très faible (le seuil se révèle plus pertinent lorsque plusieurs couples ont un score très faible).

Un dernier point est évalué uniquement pour les motifs simples nommés : **l'absence de conflits entre les noms**. Si les noms employés dans le couple *(s,c)* apparaissent dans les tables de correspondance des noms sur les mêmes lignes, il est cohérent également de les apparier. Si les noms n'apparaissent pas dans les mêmes lignes de ces tables et qu'aucun autre nom employé dans les diagrammes ne peut s'apparier avec les noms de *s* ou de *c*, alors il n'y a pas de conflit. Dans tous les autres cas, il y a un conflit et la couple est rejeté. Ce procédé est important car il permet par exemple de faciliter l'appariement des classes implicites des énoncés de l'exercice de modélisation qui peuvent avoir des noms totalement exclusifs. L'appariement de motifs nommés différemment ne peut se faire au détriment d'autres motifs candidats ayant des noms très similaires à un des motifs intervenant dans le couple mais ayant une structure altérée (par exemple en remplaçant une classe par un attribut ou inversement).

### 5.5.5 Détermination des appariements multivoques

La détermination des appariements multivoques des motifs est réalisée par une fonction *multivalentMatch* prenant en paramètres le *mapping* courant, des motifs sources et cibles non appariés. Cette fonction est capable de qualifier des appariements de cardinalité 1:n (fusion), n:1 (éclatement) et m:n (regroupement) par agrégation des couples où apparaissent des motifs communs.

*multivalentMatch(sources, targets, mapping)*

En premier lieu, les motifs non appariés sont consommés et des fusions et des éclatements sont identifiés. Les fusions sont identifiées en confrontant les motifs cibles non appariés aux motifs intervenants dans les appariements du *mapping*. Les éclatements sont identifiés en confrontant les motifs sources non appariés aux motifs intervenants dans les appariements du *mapping*. Ces deux phases sont indépendantes et un motif peut ainsi participer à une fusion et un éclatement. À chaque évaluation de deux ensembles de motifs $(s,c_1..c_N)$ ou $(s_1..s_M,c)$, un comparateur adapté est invoqué pour évaluer la similarité des motifs (cf. partie 5.4.4). La fonction *haveGoodCurrentSimilarity* est invoquée ensuite pour vérifier si les ensembles de motifs ainsi formés ont une bonne similarité globale (i.e. l'agrégation des similarités locales de chaque couple de motifs possibles). Le score de similarité complet est pris en compte. Pour les fusions, la direction est paramétrée des motifs sources vers les motifs cibles et pour les éclatements des cibles vers les sources. Enfin la cohérence de l'appariement multivoque par rapport au *mapping* courant est évalué par la fonction *isCoherent* avec comme paramètre la correspondance multivoque et le *mapping* courant. Si toutes ces conditions sont vérifiées, l'appariement multivoque est mémorisé (i.e. les motifs sources et cibles mis en correspondance et la différence multivoque décrivant l'appariement).

Les appariements multivoques précédemment identifiés ne sont pas ajoutés directement au *mapping* courant car des motifs peuvent apparaître dans une fusion et un éclatement (i.e. il y a un conflit). Deux possibilités sont envisageables dans notre proposition : **la résolution du conflit** en sélectionnant l'appariement ayant la meilleure similarité ou **l'agrégation des appariements sous forme d'un regroupement**. Dans le premier cas seul un des

appariements est retenu dans le *mapping* courant. Dans le second cas, un appariement de cardinalité m:n est constitué à partir des appariements multivoques (ils sont encapsulés dans l'appariement de regroupement).

Selon le paramétrage retenu, les appariements multivoques produits par notre méthode peuvent être limités seulement aux fusions (cardinalité globale 1:n) et aux éclatements (cardinalité globale n:1) car les regroupements (cardinalité globale m:n) sont souvent difficiles à interpréter en sortie : ils tendent en effet à mettre en correspondance globalement des motifs apparaissant localement dans un ensemble d'appariements multivoques et/ou univoques.

### 5.5.6 Identification des appariements univoques hétérogènes

Une fois que les classificateurs, les relations et les caractéristiques ont été appariés par type de manière univoque (et multivoque pour les classificateurs), des appariements univoques entre les motifs simples non appariés jusque là sont étudiés sans prendre en compte la contrainte de type. Cette phase permet d'identifier des remplacements d'éléments d'un type par des éléments d'un autre type.

Les couples sont évalués de manière gloutonne les uns après les autres jusqu'à temps qu'il n'y ait plus de couples possibles à évaluer. Tous les couples possibles à partir des motifs de départ ne sont pas évalués car certains sont rejetés dès qu'un des motifs est ajouté au *mapping*. Tout d'abord, les similarités et les différences du couple de motifs sont mesurées par un comparateur adapté : par un exemple, un comparateur d'éléments nommés pour une caractéristique et une classe (cf. partie 5.4.4). Ensuite la pertinence de ce couple est évaluée en vérifiant d'une part si les noms employés apparaissent sur la même ligne d'une des quatre tables de correspondance des noms des diagrammes et enfin par la fonction *haveGoodCurrentSimilarity* (cf. partie 5.5.4). La direction d'appariement est menée dans les deux directions en ne prenant en compte que le score simple du couple de motifs. Le score complexe n'est pas pris en compte ici car il n'est pas pertinent : la structure des éléments n'est par nature pas la même. Si le couple est retenu, il est ajouté au *mapping* courant et qualifié par une différence générale de « Remplacement ». Les motifs intervenant dans cet appariement ne peuvent pas ensuite être pris en compte dans l'évaluation d'autres appariements univoques hétérogènes.

## 5.6 Taxonomie des différences

Le terme **différence** désigne ce qui distingue deux choses, et de façon extensive ce qui distingue plusieurs choses, éventuellement prises deux à deux [Wikipedia]. Notre méthode permet de mettre en avant des différences qualifiant précisément le *mapping* des motifs des diagrammes comparés. Les différences expriment les variations du diagramme source (le diagramme de l'apprenant) par rapport au diagramme de référence (le diagramme de l'enseignant). L'ACDC produit en sortie un *mapping* de motifs appariés strictement ou partiellement. Les appariements partiels sont étiquetés avec un ou plusieurs types de différences définissant les écarts relatifs à leur agencement dans les diagrammes comparés : nous les désignons sous le terme de **différences générales**. Les appariements stricts ne signifient pas que les motifs sont identiques mais que leur structure générale est commune. Dès lors, les appariements stricts peuvent donc présenter des variations locales exprimées sous forme de **différences spécifiques**.

Tout comme les appariements, les différences peuvent être univoques lorsqu'un motif d'un diagramme est apparié avec au plus d'un motif de l'autre diagramme ou multivoques quand un motif est apparié à un ensemble de motifs. Nous avons séparé les différences relatives au formalisme des modèles comparés (les différences spécifiques) des différences propres à l'organisation et à la structuration des motifs pour ne pas être dépendant du formalisme des modèles comparés. Les différences spécifiques sont donc adaptables en fonction des propriétés spécifiques des modèles comparés. La taxonomie des différences [26] identifiables dans notre méthode est la suivante :

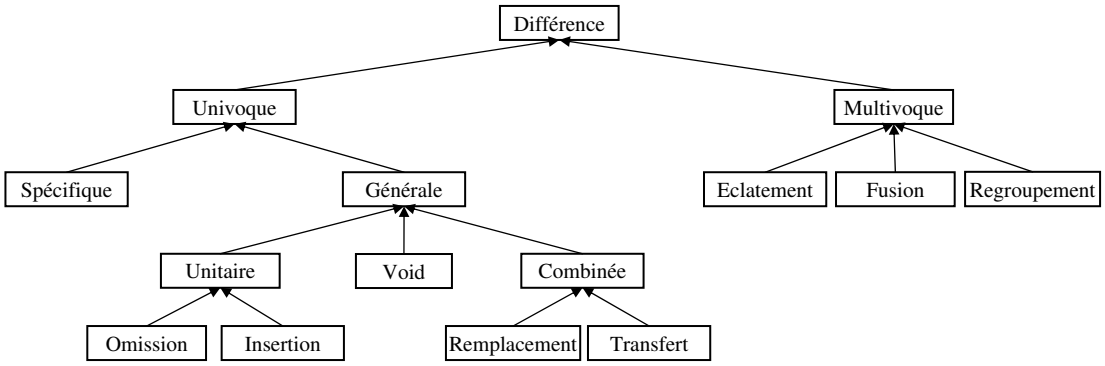



**Figure 45 : Taxonomie des différences**

### 5.6.1 Différences spécifiques

Les différences spécifiques sont relatives aux propriétés sémantiques des motifs et des modèles à comparer. Elles doivent donc être définies et adaptées en fonction des besoins d'appariement et des caractéristiques des éléments à apparier. Les différences spécifiques ont été élaborées à partir d'une étude du métamodèle des modèles à comparer. Dans le cadre des diagrammes de classes UML de niveau analyse, nous avons relevé des différences spécifiques relatives au nom, à la nature, à la visibilité, à l'abstraction, à l'agrégation et à l'orientation des relations et aux multiplicités. Les différences par rapport aux noms, à l'orientation des relations et à la nature des éléments peuvent être néanmoins généralisées à de nombreux types de modèle comme les modèles Entité-Relation, les diagrammes UML et même les graphes orientés.

L'appariement des diagrammes de la figure 46 fait ressortir quelques-unes des différences que notre méthode supporte. La schématisation en motifs complexes fait ressortir que les deux diagrammes ont une structure similaire à cette échelle (un arbre d'héritages en vert et deux chaînes d'associations respectivement en rose et en violet). Les classificateurs sont ainsi appariés un à un de manière stricte et qualifié avec le mot clé « Void ».

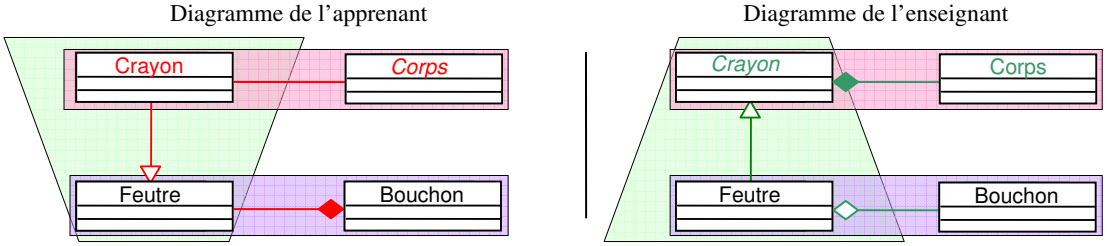



**Figure 46 : Exemples de différences spécifiques dans les diagrammes de classes UML**

[26] Nous reportons en annexe 7 la liste complète des différences identifiables dans l'ACDC.

À l'échelle de chacun des appariements de motifs simples, des différences de modélisation et de représentation sont remarquables. Les différences locales du diagramme de gauche par rapport à celui de droite sont exprimées dans la liste suivante en français et dans le formalisme que nous utilisons pour définir les différences et les appariements. Chaque ligne dans ce formalisme est de la forme :

```
{Motifs du diagramme de l'apprenant} TYPES DE DIFFÉRENCES {Motifs du diagramme de référence}
```

- Le concept « Crayon » est représenté sous forme d'une classe au lieu d'une classe abstraite :

```
{<Class>Crayon} NON_ABSTRAIT_EN_ABSTRAIT {<Class>Crayon}
```

- Le concept « Corps » est représenté sous forme d'une classe abstraite au lieu d'une classe :

```
{<Class>Corps} ABSTRAIT_EN_NON_ABSTRAIT {<Class>Corps}
```

- Une relation d'association lie les classificateurs « Crayon » et « Corps » au lieu d'une relation de composition :

```
{<Association>(Crayon---Corps)} ASSOCIATION_EN_COMPOSITION {<Association>(Crayon c--- Corps)}
```

- L'orientation de la relation d'héritage entre les classificateurs « Crayon » et « Feutre » est inversée :

```
{<Generalization>(Feutre <--- Crayon)} INVERSE {<Generalization>(Crayon <--- Feutre)}
```

- Une relation de composition lie les classificateurs « Feutre » et « Bouchon » au lieu d'une relation d'agrégation et son orientation est inversée:

```
{<Association>(Bouchon c--- Feutre)} COMPOSITION_EN_AGREGATION INVERSE {<Association>(Feutre a--- Bouchon)}
```

Il n'existe pas à proprement parler de différences spécifiques aux motifs complexes étant donné qu'ils représentent des ensembles de motifs simples. Néanmoins, les déplacements de motifs peuvent être qualifiés plus précisément localement aux motifs complexes appariés. Le concept de propagation inhérent aux hiérarchies d'héritages permet de qualifier plus précisément des déplacements de motifs vers un motif père, fils ou frère. Le concept de dérivation des chaînes d'associations permet de qualifier des déplacements vers des motifs liés. Les différences de déplacement sont quant à elles des différences générales.

### *5.6.2 Différences générales*

Les différences générales décrivent les écarts d'organisation et de structure des motifs dans les diagrammes. Elles concernent tous les types de motifs. Les variations de structure à un niveau de granularité minimal peuvent être exprimées sous forme d'**Insertion** et d'**Omission**. Les différences unitaires d'omission et d'insertion permettent d'appliquer des transformations simples sur le modèle construit par l'apprenant pour qu'il soit identique au modèle attendu : ces transformations sont respectivement « Ajouter l'élément <ELEMENT OMIS> » et « Supprimer l'élément <ELEMENT INSERE> ». L'insertion d'un motif dans le modèle construit par l'apprenant renvoie au fait qu'il n'est pas représenté dans le modèle attendu (cela peut être une sur-spécification de l'énoncé). L'omission d'un motif revient à dire qu'il n'a pas été modélisé (cela renvoie à une sous-spécification de l'énoncé).

En combinant les différences unitaires, des différences plus complexes comme le **Transfert** (déplacement d'un élément à un autre endroit) et le **Remplacement** (substitution d'un élément par un autre) peuvent être déduites. Un motif d'un diagramme peut être apparié avec plusieurs motifs de l'autre diagramme (**Éclatement**) ou vice versa (**Fusion**), et un groupe de motifs du premier diagramme peut s'apparier à un groupe de motifs de l'autre diagramme (**Regroupement**). Un même concept de l'énoncé peut être modélisé par un ou plusieurs motifs (différence multivoque d'éclatement) et inversement (différence multivoque de fusion).

Les différences sont factorisées de la plus spécifique à la plus générale. Les différences univoques générales factorisent les différences spécifiques. Une différence générale peut factoriser plusieurs différences unitaires et être elle-même factorisée par des différences d'éclatement ou de fusion. La figure 47 montre comment les différences peuvent être contenues et structurées les unes par rapport aux autres. Par exemple, une différence de regroupement est constituée de plusieurs différences de Fusion et d'Éclatement qui elles-mêmes sont constituées de plusieurs différences combinées ou Void. Les différences de regroupement masquent donc les différences de fusion et d'éclatement qui peuvent elles-mêmes masquer des différences combinées.

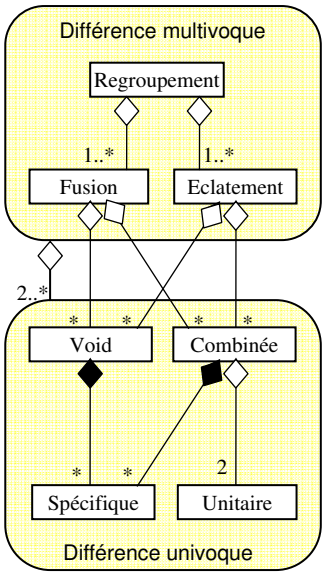



**Figure 47 : Inclusion des différences**

### 5.6.3 Exemple d'identification des différences avec ACDC

Pour illustrer plus en détail le *mapping* produit par l'ACDC, nous détaillons ici les sorties effectives d'un exemple d'appariement entre des diagrammes de classes provenant des précédentes expérimentations de l'environnement Diagram : le diagramme de l'apprenant et le diagramme de référence construit par l'enseignant sont reportés respectivement à gauche et à droite de la figure 48.

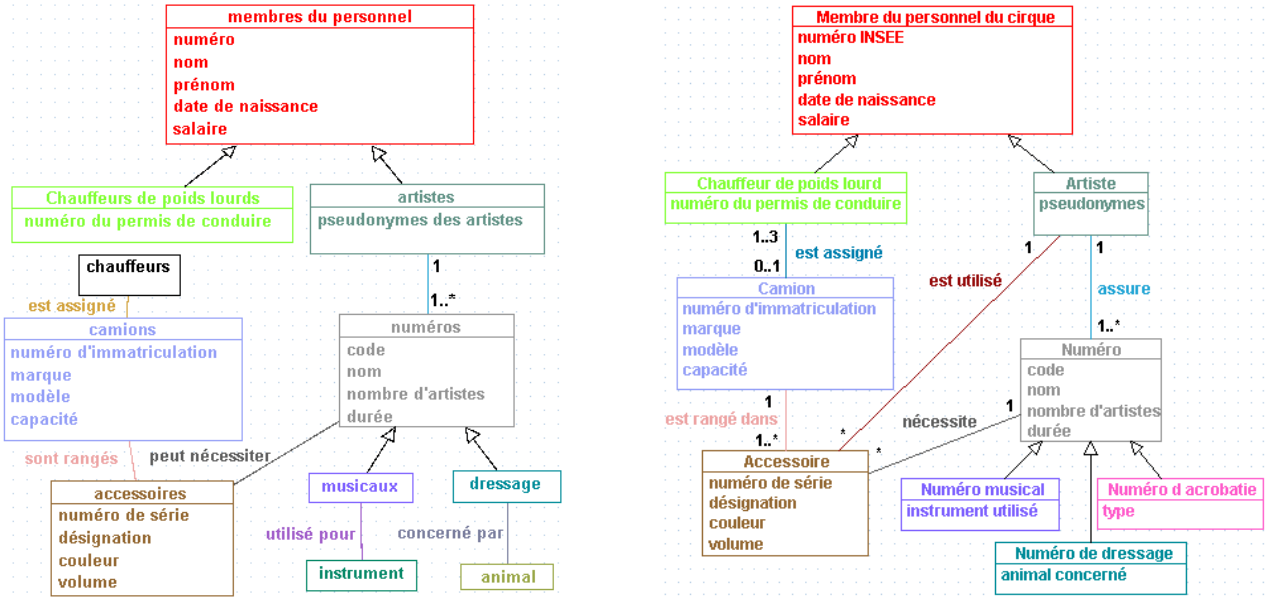



**Figure 48 : Diagramme de l'apprenant et diagramme de référence**

Dans un premier temps, les motifs des diagrammes de l'apprenant et de référence sont relevés. Il y a cinq motifs complexes et cinquante deux motifs simples dans le diagramme de l'apprenant. Seulement trois motifs complexes et quarante et un motifs simples sont présents dans le diagramme de référence (cf. figure 49).

| Motifs | Motifs du diagramme de l'apprenant | Motifs du diagramme de l'enseignant |
|---|---|---|
| Motifs complexes | 5 | 3 |
| Arbres d'héritages | 2 : AH-membres du personnel, AH-numéros | 2 : AH – Membre du personnel du cirque, AH-Numéro |
| Chaînes d'associations | 3 : CA-chauffeurs-artistes, CA-musicaux-instrument, CA-dressage-animal | 1 : CA-Chauffeurs de poids lourd |
| Motifs simples | 52 | 41 |
| Classificateurs | 11 | 9 |
| Relations | 9 | 10 |
| Caractéristiques | 32 | 22 |

**Figure 49 : Motifs identifiés dans les diagrammes de l'exemple**

Entre ces deux diagrammes de classes, trente-quatre appariements univoques stricts (entre éléments dont la structure est identique mais dont les noms peuvent différer) et neuf différences principales (appariements partiels) sont identifiés. Nous détaillons la majorité d'entre elles et donnons (dans les encadrés) les sorties correspondantes de l'ACDC. L'ensemble des sorties textuelles produites par l'ACDC sont reportées en annexe 9.

À l'échelle des motifs complexes constituants ces deux diagrammes, trois appariements sont relevés :

```
{AH-numéros CA-dressage-animal CA-musicaux-instrument} ECLATEMENT {AH-Numéro}
{AH-membres du personnel} REMPLACEMENT {AH-Membre du personnel du cirque CA-Chauffeur de poids lourd}
{CA-artistes-chauffeurs} REMPLACEMENT {CA-Chauffeur de poids lourd}
```

À l'échelle des motifs simples, huit classes sont appariées strictement entre les deux diagrammes (cf. figure 50). Les appariements stricts des relations liées à ces classes et de leurs attributs contenus ne sont pas décrits ici car ils en découlent naturellement.

| Classes du diagramme de l'apprenant | Classes du diagramme de l'enseignant |
|---|---|
| membres du personnel | Membre du personnel du cirque |
| artistes | Artiste |
| accessoires | Accessoire |
| camions | Camion |
| numéros | Numéro |
| musicaux | Numéro musical |
| dressage | Numéro de dressage |
| Chauffeur de poids lourds | Chauffeur de poids lourd |

**Figure 50 : Classes appariées strictement dans l'exemple**

L'apprenant n'a pas représenté la classe 'Numéro d acrobatie', ni son attribut 'type' et la relation d'héritage depuis cette classe. Trois omissions sont repérées par l'ACDC :

```
OMISSION {Numéro d acrobatie}
OMISSION {Numéro d acrobatie::type}
OMISSION {(Numéro<--Numéro d acrobatie)}
```

L'apprenant a créé une classe 'instrument', reliée à la classe 'musicaux', alors que l'enseignant a défini l'instrument de musique comme un attribut de la classe 'Numéro musical'. ACDC détecte tout d'abord l'insertion d'une classe et l'omission d'un attribut : ces deux différences sont factorisées et reconnues dans le *mapping* final comme un appariement hétérogène d'un attribut et d'une classe (lignes 1 à 3). Une précision indique que ce remplacement porte sur deux éléments de nature différente (lignes 4). L'insertion d'une relation entre la nouvelle classe et la classe 'musicaux' est identifiée (ligne 5). De la même manière, l'apprenant a introduit une classe 'animal' au lieu d'un attribut 'animal concerné' de la classe 'Numéro de dressage', et des différences similaires sont repérées.

```
{instrument} REMPLACEMENT {Numéro musical::instrument utilisé}
             OMISSION     {Numéro musical::instrument utilisé}
{instrument} INSERTION
{instrument} NATURE_INCOMPATIBLE {Numéro musical::instrument utilisé}
{utilisé pour(musicaux---instrument)} INSERTION
```

La notion de 'chauffeur' est représentée par deux classes dans le diagramme de l'apprenant : une classe 'chauffeurs ' et une classe 'Chauffeurs de poids lourds'. ACDC apparie chacune de ces classes à la classe 'Chauffeur de poids lourd' du diagramme de référence, et considère cette différence comme l'éclatement d'une classe en deux classes.

```
{chauffeurs chauffeurs de poids lourds} ECLATEMENT {Chauffeur de poids lourd}
```

Enfin, l'apprenant n'a pas représenté de relation d'association entre les classes 'Artiste' et 'Accessoire' :

```
OMISSION {utilisé(Artiste---Accessoire)}
```

## 5.7 Paramétrage et complexité de la méthode proposée

Nous avons précisé dans les parties précédentes que notre méthode peut être adaptée en fonction des besoins d'appariement. Pour cela, il faut définir clairement la nature des appariements recherchés. Le paramétrage de la mesure de similarité et des apparieurs donne tout son sens au *mapping* recherché et constitue donc une difficulté de notre

approche. Nous présentons dans cette dernière partie un exemple de paramétrage de l'ACDC et ensuite une évaluation de la complexité de notre méthode.

### 5.7.1 Paramétrage de la méthode

Pour paramétrer notre méthode, nous avons étudié conjointement la sémantique, la structure définie dans le métamodèle d'UML et comparé manuellement un jeu de 82 diagrammes de classes UML construits en situation réelle d'apprentissage par des étudiants. Ces exercices sont de difficulté et de taille croissante et couvrent la majeure partie des difficultés rencontrées dans les diagrammes de classes de niveau analyse (cf. partie 7.2).

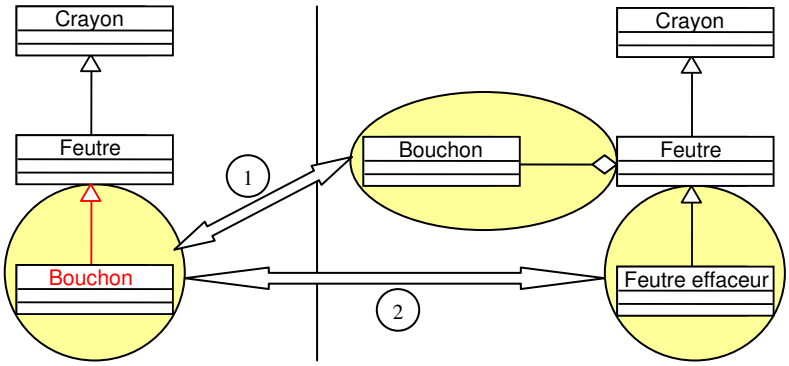



**Figure 51 : Exemples d'appariements possibles pour deux diagrammes de classes UML**

La figure 51 expose deux correspondances différentes qui pourraient être identifiées entre les deux diagrammes présentés en paramétrant différemment notre méthode. Le premier appariement (numéroté 1 dans la figure) est obtenu en donnant un poids fort aux critères de similarité des espaces de nommage dans le comparateur d'éléments nommés. À contrario, les critères des types des relations dans le comparateur de relations ont un poids faible pour ne pas entraver l'appariement en cas de remplacement d'une relation d'un type par un autre. Dans cette correspondance, les classes nommées « Bouchon » sont appariées car leurs noms sont identiques. La relation d'agrégation devant lier la classe « Bouchon » à la classe « Feutre » est remplacée par une relation d'héritage. La classe « Feutre effaceur » et la relation d'héritage devant lier cette dernière à la classe « Feutre » n'ont pas été représentées. Le second appariement est obtenu à partir d'un paramétrage focalisé sur le maintien des structures et de la sémantique des relations et des hiérarchies. Les dimensions relatives à la nature des relations et à la propagation dans les hiérarchies (i.e. les scores propagés des motifs contenus et liés) ont un poids fort alors que le critère de similarité des espaces de nommage est beaucoup plus faible. La classe « Bouchon » est la classe fille de la classe « Feutre » dans le diagramme de gauche. La classe « Feutre effaceur » est la classe fille de la classe « Feutre » dans le diagramme de droite. Les classes « Bouchon » et « Feutre effaceur » ont le même comportement, les mêmes relations et sont appariées sous ce point de vue structurel. La classe « Bouchon » et la relation d'agrégation liée à la classe « Feutre » n'ont pas été représentées dans le diagramme de gauche.

Nous avons paramétré notre méthode pour qu'elle retienne le premier appariement, étant donné que les noms donnent un sens important au diagramme de classes modélisé. Néanmoins les aspects structurels des diagrammes ne sont pas négligés et permettent de repérer les cas de déplacement d'éléments ou de classes implicites. C'est l'aspect structurel qui permet d'apparier correctement les classes implicites si la structure des deux diagrammes n'est pas trop altérée.

### 5.7.2 Évaluation de la complexité

La complexité de calcul de notre méthode dépend de la taille des diagrammes à apparier, du nombre de comparaisons réalisées, et de la « distance » entre ces diagrammes : des diagrammes très différents requièrent plus de tests et de comparaisons. La complexité du processus complet d'appariement est la somme des complexités requises par chaque phase : l'évaluation des similarités et des différences locales aux motifs des diagrammes, l'appariement final des motifs complexes et l'appariement final des motifs simples.

La complexité de la phase de mesure des similarités et de quantification des différences est directement proportionnelle au nombre de couples de motifs comparés, triés par type (les motifs complexes, les classificateurs, les relations et les caractéristiques comparés par paires). La comparaison est d'autant plus longue qu'il y a de motifs dans chacun des diagrammes à comparer. Néanmoins, les comparaisons ne sont réalisées qu'une seule fois pour chaque couple de motifs car les résultats sont stockés pour pouvoir être réutilisés à chaque interrogation. Par exemple la comparaison de deux diagrammes composés de dix classificateurs, douze relations et sept caractéristiques répartis dans trois motifs complexes requiert deux cents quatre-vingt dix-neuf comparaisons, c'est-à-dire cent comparaisons au niveau des classificateurs, cent quarante-quatre au niveau des relations, quarante-neuf au niveau des caractéristiques et enfin six au niveau des motifs complexes.

Chaque phase d'appariement est gloutonne (les algorithmes glouton sont de complexité polynômiale) : dans le meilleur des cas, seuls des appariements univoques sont calculés avec une complexité minimale de $O(n)$. Dans le pire des cas, tous les appariements univoques et multivoques possibles sont calculés, avec une complexité maximale de $O(n*m)^2$ où $n$ et $m$ sont les nombres de motifs respectivement dans les diagrammes de l'apprenant et de l'enseignant. Notre proposition limite le calcul des appariements multivoques et ne peut atteindre cette complexité maximale théorique. De plus, le calcul de tous les appariements multivoques n'a pas d'intérêt étant donné que la majorité d'entre eux sont incohérents avec le *mapping* courant.

La précision du résultat et la complexité du processus d'appariement doivent être mises en balance : le *mapping* proposé ne doit pas seulement être pertinente mais doit aussi être rapide pour les besoins de communication en temps réel avec l'apprenant (i.e. des rétroactions pédagogiques) [Giunchiglia *et al.* 2007]. La rapidité est un paramètre fondamental car notre méthode est appliquée dans Diagram où le système doit interagir de manière synchrone avec l'apprenant qui construit son diagramme à partir de l'énoncé textuel. C'est pourquoi nous avons opté dans notre proposition pour un processus d'appariement glouton car il produit un appariement correct de manière rapide. À contrario, la phase de mesure des similarités et des différences à toutes les échelles des modèles peut être un facteur limitant lorsque les diagrammes comportent beaucoup d'éléments car cette dernière repose actuellement sur une comparaison complète par type d'éléments.

# CHAPITRE 6

# Application de la méthode d'appariement de diagrammes au diagnostic

# PLAN DU CHAPITRE

Notre méthode d'appariement de diagrammes (cf. chapitre 5) qualifie des différences relatives à la structure et à la sémantique des deux diagrammes comparés (un diagramme de l'apprenant et un diagramme de référence). Ces différences ne sont pas exploitables en l'état pour des besoins pédagogiques tels que le diagnostic[27] et la production de rétroactions pédagogiques car elles ne constituent pas nécessairement des erreurs que commet l'apprenant. La production de rétroactions pédagogiques pour l'apprenant lors de l'activité de modélisation a pour objectif dans Diagram d'encourager la réflexion de l'apprenant sur son activité de modélisation pour qu'il corrige lui-même son diagramme. Cela nécessite d'analyser les résultats issus de l'appariement des deux diagrammes et de relever les différences nécessitant une intervention de la part du système. Les différences peuvent notamment être filtrées ou/et regroupées pour ne présenter que celles jugées pertinentes pour des besoins pédagogiques de rétroactions à l'apprenant.

L'identification précise des erreurs de syntaxe n'est pas gérée par la méthode car c'est un problème de vérification de conformité (cf. partie 2.1.1) qui ne nécessite pas de comparer les deux diagrammes. Leur recherche peut donc être réalisée à part par un module de vérification de la cohérence que nous définissons dans ce chapitre.

Dans la suite de ce chapitre, nous allons exposer une application de notre méthode d'appariement au diagnostic des diagrammes de classes UML de niveau analyse construits par l'apprenant dans l'environnement Diagram. Son principe général est le suivant : les différences relevées par notre méthode, qui sont qualifiées dans la suite du chapitre de **différences structurelles** et les erreurs syntaxiques sont filtrées, organisées et converties en **différences pédagogiques** directement utilisables pour fournir des rétroactions à l'apprenant. Les différences pédagogiques ainsi que les rétroactions sont issues des travaux de thèse de Mathilde Alonso [Alonso 2009]. Nous avons été amené à adapter les résultats produits par notre méthode pour le diagnostic du diagramme de classes UML construit par l'apprenant et la production de rétroactions pédagogiques.

Nous présentons dans la suite de cette partie tout d'abord notre méthode de vérification de la cohérence des diagrammes produits par l'apprenant et les erreurs relatives à la syntaxe des diagrammes de classes UML de niveau analyse. Ensuite, nous explicitons la manière dont les erreurs syntaxiques et les différences structurelles sont utilisées dans Diagram et quelles sont les rétroactions produites pour encourager l'apprenant à réfléchir sur sa production.

## 6.1 Vérification de la cohérence du diagramme de l'apprenant

La **vérification de la cohérence** et du **respect de la syntaxe** d'un modèle permet de repérer des erreurs syntaxiques liées à des problèmes de connaissance et de représentation des éléments modélisés (cf. partie 2.1.1). Les erreurs syntaxiques découlent ainsi d'une utilisation non-conforme d'un élément de modélisation par rapport à la syntaxe et au formalisme du langage de modélisation utilisé. Il s'agit d'un problème de représentation (« la forme ») et non de modélisation (« le fond ») ; la représentation doit être conforme au langage de représentation alors que la modélisation doit être conforme aux spécifications. Une erreur de syntaxe peut également renvoyer au fait de savoir ce qu'il faut représenter mais de ne pas savoir comment le représenter. Une erreur de syntaxe a des conséquences sur le

[27] Le diagnostic est entendu ici comme la faculté du système à évaluer les réponses de l'apprenant (i.e. le diagramme construit par l'apprenant).

sens véhiculé par le modèle. De par leur nature et du fait qu'elles sont totalement indépendantes du modèle idéal, les erreurs syntaxiques sont identifiables en observant uniquement le modèle construit par l'apprenant.

Dans le contexte des diagrammes UML, les erreurs syntaxiques enfreignent le métamodèle d'UML et ses règles de cohérence associées. Les erreurs syntaxiques pouvant être rencontrées dans un diagramme de classes UML sont nombreuses et déductibles automatiquement à partir des règles de cohérence relevées précisément dans [Seuma Vidal *et al.* 2005]. Dans notre cas, seules les règles propres à la cohérence interne d'un diagramme nous intéressent, étant donné qu'un seul diagramme peut être construit simultanément dans l'environnement Diagram. Nous reportons en annexe 3 un extrait de ce document et commentons ici les principales règles pour illustrer plus concrètement les notions d'incohérences et d'erreurs syntaxiques internes à un diagramme de classes UML 2.x :

- **Absence du nom d'un élément nommé :** par exemple, une classe, un attribut ou une opération (encapsulé dans une classe) doivent être obligatoirement nommées (règles 44, 76 et 77).
- **Noms de plusieurs éléments non uniques :** les classes d'un même diagramme doivent toutes avoir un nom différent ; les attributs possédés par une même classe doivent avoir des noms qui permettent de les différencier (règle 4).
- **Non respect de la syntaxe d'un élément :** les multiplicités, les propriétés, les opérations et les contraintes ont une syntaxe précise dans UML, décrite sous forme d'expressions régulières (règles 13, 23, 32, 55, 56, 76, 77).
- **Mise en relation incomplète d'éléments :** une relation ne peut pas être liée à un seul élément (si elle n'est pas réflexive) mais à au moins deux éléments suivant sa nature (règles 132 et 175).
- **Mise en relation incohérente de plusieurs éléments** : une relation entre plusieurs éléments n'est pas autorisée si au moins un des éléments ne supporte pas ce type de lien (règles 130 et 174).
- **Cycle d'héritage interdit** : un classificateur ne doit pas être à la fois un ancêtre et un descendant d'un même classificateur dans une hiérarchie de généralisation (règle 131).
- **Classe d'association réflexive interdite :** une classe d'association ne peut pas être définie entre elle-même et quelque chose d'autre. Il ne faut pas non plus que la classe d'association soit un parent ou un enfant d'une des classes mises en relation (règle 176).

L'exemple de la figure 52 illustre le fait que la vérification de la cohérence et de la syntaxe du diagramme de l'apprenant (diagramme de gauche) peut être traitée indépendamment de la comparaison et de l'appariement des constituants de deux diagrammes. Dans le diagramme de gauche, l'erreur syntaxique « Cycle d'héritage entre les classes C1, C3 et C4 » peut être relevée de manière indépendante même si elle est une conséquence[28] des deux différences suivantes : d'une part, l'inversion du sens de la relation d'héritage entre la classe « C3 » et la classe « C1 » et d'autre part le remplacement de la relation d'association entre les classes « C3 » et « C4 » par une relation d'héritage. Enfin l'erreur syntaxique « Absence de nom pour une classe » n'a pas de lien direct avec la différence de remplacement de la classe « C2 » par la classe sans nom qui peut être qualifiée du point de vue de la structure commune des deux diagrammes à comparer.

[28] Un cycle d'héritage est interdit quelque soit le diagramme analysé. L'appariement du diagramme de l'apprenant et de la référence n'identifie pas cette erreur de syntaxe en elle-même mais il permet de découvrir les différences engendrant cette erreur de syntaxe dans le diagramme de l'apprenant. Un remplacement de type et une inversion sont dans ce contexte les causes possibles du cycle d'héritage. De plus, un cycle d'héritage peut apparaître dans d'autres situations, par exemple un ajout de relation d'héritage dans le diagramme de l'apprenant.

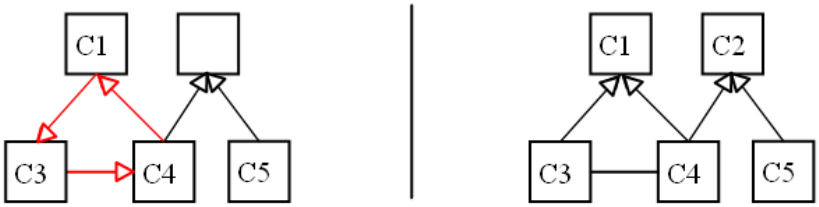



**Figure 52 : Exemple d'erreurs syntaxiques et de différences repérées**

Les règles de cohérence et de syntaxe citées précédemment doivent être contrôlées pour savoir si le diagramme est cohérent et syntaxiquement correct. Ce contrôle peut mis en place en partie ou complètement au niveau de l'interface de l'éditeur graphique de modèles lors de la création et de la modification d'un élément graphique en interdisant directement certaines constructions erronées. Dans ce cas de figure, les erreurs de syntaxe sont interceptées à l'interface et n'apparaissent pas dans le diagramme final. La vérification de la cohérence peut être aussi réalisée à la demande en se focalisant sur une partie ou l'intégralité du diagramme déjà construit.

Dans notre cas, l'environnement Diagram ne contrôle pas à l'interface toutes les constructions erronées car l'éditeur de modèles intégré est prévu pour s'adapter à tout type de graphes et d'éléments : l'interface d'édition est générique et repose sur JGraph [JGraph] qui n'est pas spécifique aux diagrammes UML. Elle ne contrôle pas la syntaxe d'UML et autorise par exemple la création de n'importe quel type de liens graphiques (relation, lien d'attachement, lien de classes d'association) entre n'importe quel type d'élément, y compris les liens eux-mêmes.

Il est donc nécessaire de mettre en place un système de validation efficace et complet capable d'identifier et d'expliciter les erreurs syntaxiques apparaissant dans le diagramme UML de l'apprenant. Ce problème a été traité dans de nombreux éditeurs de diagrammes UML et la solution se limite souvent à une simple vérification et une validation relative à la sémantique du modèle objet. Dans une optique de réutilisation de la mécanique de validation syntaxique proposée par ces éditeurs, un problème subsiste : il n'est pas possible d'accéder et d'utiliser librement les sources des éditeurs commerciaux tels que Rose, Together ou Objecteering. Nous nous sommes ainsi tournés vers le projet UML2, défini et développé par la communauté *open-source* Eclipse [Eclipse]. Le formalisme proposé pour représenter les diagrammes est en conformité avec le standard actuel UML 2.x (par exemple, les diagrammes sont représentables et manipulables en XMI). De plus, il propose une vérification de la cohérence et de la syntaxe similaire aux règles de [Seuma Vidal *et al.* 2005].

La vérification de la syntaxe et de la cohérence du diagramme de l'apprenant est réalisée une fois qu'il est créé graphiquement dans Diagram. Elle s'effectue en deux étapes dans notre proposition : tout d'abord, lors de la conversion du modèle graphique, et ensuite, sur le diagramme converti. La souplesse d'édition de l'éditeur de Diagram impose de mettre en place une première phase de vérification de la syntaxe lors de la conversion du modèle graphique en diagramme conforme au formalisme UML 2.x : des erreurs sont identifiées lors de cette phase au niveau des constructions graphiques ne pouvant être converties en l'état car elles ne peuvent pas être représentées dans le format retenu (elles n'y ont simplement pas d'équivalent). Elles concernent essentiellement le non respect de la syntaxe d'un élément et la mise en relation incohérente de plusieurs types d'éléments. Par exemple, une relation d'association représentée entre deux autres relations ne peut pas être convertie car il n'est pas autorisé intrinsèquement de faire ce lien dans le format cible conforme au métamodèle d'UML. Une fois le modèle converti, il est analysé pour identifier les autres problèmes de cohérence et de syntaxe. Cette vérification repose sur le composant UML2.

Les erreurs syntaxiques (cf. figure 53) sont qualifiées avec un **code d'erreur principal** définissant de manière générale l'erreur et un **code secondaire** précisant en détail l'erreur. Le message d'erreur et l'élément graphique correspondant sont associés à l'erreur. Si un élément possède un équivalent dans le diagramme UML converti, il est également encapsulé dans l'erreur. Un même élément peut être qualifié avec plusieurs erreurs syntaxiques. Par exemple, si la multiplicité d'une extrémité d'une association est notée « 1..0 » alors les erreurs 4.7 et 4.8 seront identifiées lors de la conversion de la multiplicité.

```
1 Erreur de lien ou de relation : élément ne pouvant être lié à un autre élément
     1.1  erreur relative à une relation uml2 : un élément différent d'un classificateur ne peut
          être lié à une relation (classificateur --- classificateur)
     1.2  erreur relative à un lien d'attachement (commentaire --- autre élément uml2)
     1.3  erreur relative à un lien de classe d'association (association --- classe)

2 Erreur de contenance : élément ne pouvant être contenu par un autre élément
     2.1  caractéristique (attribut ou opération) ne pouvant être contenue par un élément différent
          d'un classificateur

3 Elément incomplet
     3.1  classificateur non nommé
     3.2  relation incomplète (liée à 0 ou 1 classificateur)
     3.3  lien d'attachement incomplet
     3.4  classe d'association incomplète
     3.5  caractéristique non nommée
     3.6  multiplicité non remplie (cas particulier d'omission)

4 Elément mal formé
     4.1  relation mal formée
     4.2  lien d'attachement mal formé
     4.3  classe d'association mal formée
     4.4  multiplicité non représentée par un intervalle ou un nombre
     4.5  borne basse d'une multiplicité négative
     4.6  borne haute d'une multiplicité négative
     4.7  borne haute d'une multiplicité inférieure à sa borne basse
     4.8  borne haute d'une multiplicité égale à 0
     4.9  borne haute d'une multiplicité sous forme d'intervalle égale à sa borne basse
     4.10 multiplicité négative
     4.11 multiplicité égale à 0

5 Erreurs de cycles
     5.1  cycle d'héritage
     5.2  classe d'association réflexive

6 Avertissements divers
     6.1  redondance des noms des éléments
```

**Figure 53 : Erreurs et avertissements concernant la syntaxe des diagrammes de classes d'analyse**

## 6.2 Utilisation d'ACDC dans Diagram

Le système ACDC produit une liste d'appariements qualifiés par des différences entre le diagramme de l'apprenant et le diagramme de référence, suivant la **taxonomie des différences structurelles** (notée TDS). L'utilisation de ces différences est focalisée sur la production de rétroactions à visée pédagogique fournies à l'apprenant pour l'encourager à réfléchir sur sa proposition. Ces différences ne sont pas directement exploitables pour produire des rétroactions à visée pédagogique car elles sont orientées essentiellement sur la structure des éléments des diagrammes comparés et elles ne sont pas hiérarchisées lorsqu'elles sont trop détaillées pour des besoins pédagogiques. Il est nécessaire de les convertir en différences selon une autre taxonomie, la **taxonomie des différences au sens pédagogique** (notée TDP), élaborée du point de vue pédagogique et définie dans le contexte de la thèse de Mathilde Alonso [Alonso 2009]. Une fois cette conversion effectuée, un message de rétroaction peut être élaboré pour traiter chaque différence.

Les travaux de Mathilde Alonso se concentrent sur l'interaction entre l'environnement Diagram et l'apprenant. Elle a notamment défini dans ce cadre la TDP, les rétroactions associées et les tables de conversion de la TDS vers la TDP permettant ensuite de produire les rétroactions pédagogiques. Nous nous sommes occupés d'implanter la TDP et la conversion de la TDS en TDP. Nous avons vérifié et adapté conjointement avec Mathilde Alonso la conversion des différences par rapport aux besoins pédagogiques exprimés dans Diagram.

### *6.2.1 Taxonomie des Différences Pédagogiques*

La taxonomie des différences au sens pédagogique a été construite à partir de la comparaison manuelle de diagrammes de l'apprenant avec un diagramme de référence pour deux exercices représentatifs de notre corpus. À partir de la liste exhaustive de différences relevées, une taxonomie de différences « simples », qui sert de base à la construction des rétroactions pédagogiques, a été identifiée. En effet, chaque différence simple permet la construction d'un message de rétroaction. Les différences simples sont regroupées en huit catégories :

- **Omission d'un élément :** l'apprenant a omis un élément du diagramme de référence.
- **Ajout d'un élément :** l'apprenant a ajouté dans son diagramme un élément qui ne figure pas dans le diagramme de référence.
- **Transfert d'un élément :** un élément a été transféré dans une autre partie du diagramme. Par exemple une relation entre deux classes « A » et « B » dans le diagramme de référence est déplacée par l'apprenant entre les classes « A » et « C ».
- **Dédoublement d'un élément :** un élément du diagramme de référence est représenté dans le diagramme de l'apprenant par plusieurs éléments du même type.
- **Fusion d'éléments :** plusieurs éléments d'un même type sont représentés dans le diagramme de l'apprenant par un seul élément.
- **Représentation erronée :** un élément du diagramme de référence est représenté dans le diagramme de l'apprenant mais sous une autre forme. Par exemple, une classe est remplacée par un attribut, une composition est remplacée par une association, etc.
- **Inversion du sens d'une relation :** le sens d'une relation orientée (héritage, agrégation ou composition) a été inversé par l'apprenant.
- **Multiplicité erronée :** les multiplicités d'une relation dans le diagramme de l'apprenant comportent des différences avec celles du diagramme de référence.

Dans les diagrammes des apprenants, certaines différences s'accompagnent souvent d'autres différences simples. Par exemple, la différence « omission d'une classe » peut entraîner les différences « omission d'une relation » ou « transfert d'une relation », car si l'apprenant oublie de représenter une classe, cela a pour effet l'omission des relations liées à cette classe ou leur transfert vers une autre classe. Dans ce cas, il est préférable de produire une seule rétroaction générale sur ce groupe de différences, plutôt que de produire chacune des rétroactions élémentaires. Un ensemble de quinze différences « composées » a été défini de manière à les appréhender globalement. Une différence composée est constituée d'une différence principale et d'un ensemble de différences associées. Par exemple, la

différence composée « omission d'une classe et des éléments associés » est formée de la différence principale « omission d'une classe » et de différences associées telles que « omission d'un attribut de cette classe », « omission d'une relation portant sur cette classe ». Une fois que toutes les différences simples ont été identifiées, l'accent est mis en premier lieu sur le repérage de combinaisons d'erreurs. Pour chaque différence principale, une recherche de la présence d'une ou plusieurs différences associées est lancée. Les différences combinées obtenues sont prioritaires sur les différences simples qui les composent, et donnent lieu à une rétroaction spécifique. Les différences simples restantes, qui se sont produites de manière isolée, donnent lieu à une rétroaction séparée.

### 6.2.2 Transcription des différences structurelles en différences pédagogiques

Après avoir établi la taxonomie des différences simples et composées, l'intérêt a été concentré sur la correspondance entre les deux taxonomies, TDS et TDP. En effet, le diagnostic produit une liste de différences en terme d'appariement de motifs structurels, et ces différences ne sont pas nécessairement utilisables directement pour construire les rétroactions. Pour chaque diagramme réalisé par un étudiant, les sorties du diagnostic et les différences simples identifiées manuellement ont été comparées afin de vérifier leur concordance. Le tableau général de correspondance entre les deux taxonomies est le suivant :

| Différence structurelle de la TDS | Différence pédagogique de la TDP |
|---|---|
| multivoque d'éclatement | dédoublement |
| multivoque de fusion | fusion |
| multivoque de regroupement | *pas de correspondance* |
| univoque spécifique | représentation erronée, inversion du sens d'une relation, multiplicités |
| univoque unitaire d'omission | omission |
| univoque unitaire d'insertion | ajout |
| univoque combinée de transfert | transfert |
| univoque combinée de remplacement | *pas de correspondance* |
| univoque de type void | *pas de correspondance* |

**Figure 54 : Correspondance entre les deux taxonomies de différences**

Les différences multivoques d'éclatement et de fusion correspondent respectivement à des différences de dédoublement et de fusion. Une différence de regroupement correspond au cas où un groupe de motifs du premier diagramme comparé est apparié à un autre groupe de motifs du second diagramme. Cette différence apporte une information sur les structures des diagrammes comparés mais semble difficile à exploiter à des fins pédagogiques. Aucune correspondance de cette différence n'a été trouvée dans la taxonomie TDP. La catégorie des différences univoques spécifiques regroupe les différences relatives aux propriétés et à la sémantique des motifs. Dans la TDP, les cas de représentation erronée (classe au lieu d'attribut ou inversement, type de relation erroné, etc.), et également le cas d'inversion du sens d'une relation se retrouvent. Les différences unitaires d'omission et d'insertion, lorsqu'elles ne sont pas factorisées par d'autres différences, correspondent aux différences d'omission et d'ajout. De même, une différence de transfert qui n'est pas factorisée par une différence multivoque, dans la TDS correspond à un transfert dans la TDP. Les différences non factorisées de remplacement et de type *Void* expriment le fait que les motifs sont appariés strictement et c'est pour cela qu'elles n'ont donc pas de correspondance dans la TDP.

### 6.2.3 Production des rétroactions pour l'apprenant dans Diagram

Dans la méthode ACDC (cf. chapitre 5), l'appariement du diagramme de l'apprenant avec le diagramme de référence produit une liste de différences. Celles-ci ne traduisent pas nécessairement une erreur, mais peuvent suggérer que l'étudiant a effectué un mauvais choix de modélisation. Les messages de rétroactions dans Diagram ne sont donc pas des « messages d'erreur » classiques. Ces messages visent à attirer l'attention de l'étudiant sur certaines parties de son diagramme afin de solliciter les fonctions de régulation métacognitive. Le diagnostic s'effectuant une fois la tâche réalisée, c'est principalement la fonction de vérification qui est visée. Dans un petit nombre de cas, cependant, la différence observée suggère plutôt que l'apprenant ne maîtrise pas certaines notions du langage UML : la rétroaction se fait alors au niveau cognitif, par exemple sous forme de rappel de cours.

La méthode de diagnostic par appariement développée dans Diagram nécessite que le diagramme de l'apprenant soit complet ou quasi-complet. Sur un diagramme incomplet, les éléments non encore modélisés ne peuvent pas être distingués des éléments omis, et le diagnostic les considérerait comme des omissions. La phase d'analyse du diagramme a donc été intégrée à l'issue de l'étape de relecture (cf. partie 2.4.2). L'apprenant peut ensuite revenir à la phase de construction pour modifier son diagramme en fonction des rétroactions reçues.

Selon les cas, une différence repérée entre les diagrammes peut traduire une erreur ou bien une simple variante de représentation de l'énoncé. La forme des messages de rétroactions doit donc être modulée selon le degré de certitude sur la présence d'une erreur. En s'appuyant sur la classification proposée par [Lemeunier 2000], trois formes d'intervention ont été retenues : **indiquer**, **questionner** et **proposer**. « Indiquer » consiste à attirer l'attention de l'étudiant sur une partie du diagramme, ou sur la manière dont il a représenté une notion de l'énoncé. Cela peut consister en une reformulation textuelle d'un élément du diagramme, visant à faire percevoir à l'apprenant que la représentation qu'il a choisie est inadéquate. « Questionner » consiste à interroger l'apprenant sur les caractéristiques de son diagramme. Les questions sont de type binaire (réponse oui/non) et incitent l'apprenant à vérifier mentalement que le diagramme satisfait une propriété donnée. La modalité « Proposer », la plus directe, consiste à suggérer la correction à apporter au diagramme : elle ne s'applique que lorsque la présence d'une erreur est quasi-certaine.

Selon les types de différences, une ou plusieurs modalités d'intervention seront appropriées. Ainsi, en cas d'ajout ou d'omission d'éléments, qui ne constituent pas nécessairement des erreurs, les modalités « Indiquer » et « Questionner » sont utilisées. Dans certains cas de représentation incorrecte d'une relation (par exemple, un héritage au lieu d'une composition), la modalité « Proposer » peut également être utilisée car la présence d'une erreur est très probable.

Au cours de l'interaction, les interventions sont graduées de la plus générale (« Indiquer ») à la plus précise (« Proposer ») : pour chaque différence observée, le message le plus général est affiché, puis l'apprenant peut faire apparaître le message de niveau plus spécifique s'il le souhaite.

### 6.2.4 Exemple de production des rétroactions pour l'apprenant dans Diagram

Nous présentons dans cette section un exemple complet afin d'illustrer le processus de production des rétroactions à partir du diagramme d'un apprenant qui repose sur notre module de diagnostic. L'exercice de cet exemple est celui que nous avons présenté dans la partie 1.4.1. Le diagramme de l'apprenant reporté dans la figure 55 est fictif : nous y avons regroupé des différences variées que nous avons rencontrées en analysant de réels diagrammes produits par des apprenants pour cet exercice. La solution de référence de la figure 55 est celle exprimée au niveau analyse (sans les méthodes et les attributs supplémentaires). Nous n'avons pas reporté les multiplicités dans ces diagrammes pour ne pas surcharger inutilement l'exemple.

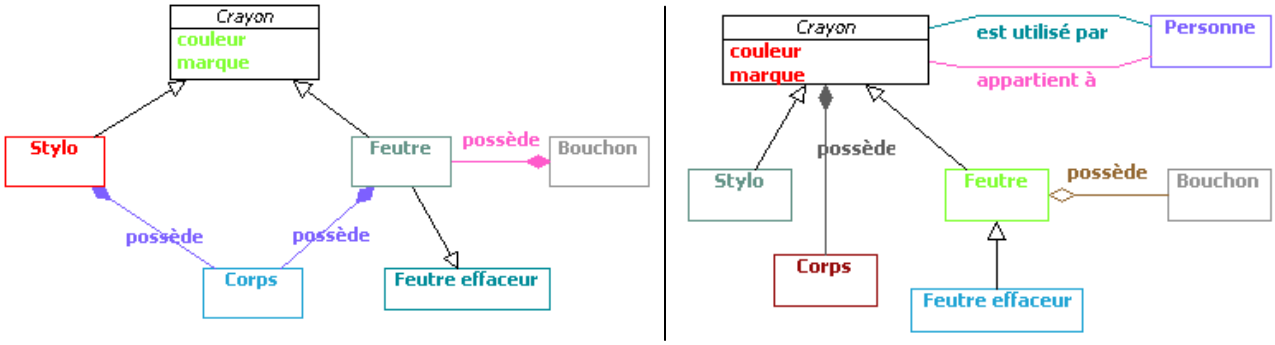



**Figure 55 : Diagramme de l'étudiant (à gauche) et diagramme de référence (à droite)**

En premier lieu, le diagramme produit par l'apprenant et le diagramme de référence sont comparés et leurs motifs sont appariés par l'ACDC. Treize différences structurelles ainsi que treize appariements stricts (*Void*) sont produits.

| Différences TDS | Différences simples TDP | Différences composées TDP |
|---|---|---|
| OMISSION {Personne} | Omission d'une classe | (A) Omission d'une classe et des éléments liés à cette classe |
| OMISSION {appartient à (Personne---Crayon)} | Omission d'une relation | |
| OMISSION {est utilisé par (Personne---Crayon)} | Omission d'une relation | |
| {possède (Corps---Feutre) possède (Corps---Stylo)} ECLATEMENT {possède (Corps---Crayon)} | Dédoublement d'une relation | (B) Dédoublement et transfert d'une relation |
| {possède (Corps---Feutre)} TRANSFERT VERS_FILS {possède (Corps---Crayon)} | Transfert d'une relation | |
| {possède (Corps---Stylo)} TRANSFERT VERS_FILS {possède (Corps---Crayon)} | Transfert d'une relation | |
| {possède (Feutre---Bouchon)} INVERSE {possède (Bouchon---Feutre)} | Inversion du sens d'une relation | (C) Représentation erronée d'une relation et inversion de sens |
| {possède (Feutre---Bouchon)} COMPOSITION_EN_AGREGATION {possède (Bouchon---Feutre)} | Type erroné de relation | |
| {Feutre effaceur-->Feutre} INVERSE {Feutre<--Feutre effaceur} | (D) Inversion du sens d'une relation | |

**Figure 56 : Mise en correspondance des différences TDS et TDP dans l'exemple**

Parmi les treize différences, neuf sont mises en correspondance avec des différences pédagogiques, les autres peuvent être ignorées car elles sont factorisées par des différences de niveau supérieur. Trois ensembles de différences simples sont reconnus comme des différences composées (A, B et C), la dernière différence simple est isolée (D).

Pour chacune des ces quatre différences pédagogiques, un message de rétroaction est produit selon les modalités d'intervention définies dans la partie 6.2.3. Dans l'exemple, l'apprenant a omis la classe « Personne », ainsi que les relations « est utilisé par » et « appartient à » entre les classes « Crayon » et « Personne ». Le diagnostic repère trois différences structurelles qui correspondent à trois différences simples dans la taxonomie TDP. Ces différences simples forment une différence pédagogique composée, qui exprime l'omission d'une classe et des éléments liés à cette classe (différence A). À partir de cette différence, une rétroaction pédagogique est obtenue selon deux modalités :

*Indiquer* : Le diagramme est incomplet. Un concept important n'a pas été représenté.
*Questionner* : As-tu représenté le concept de 'Personne' ?

Dans le diagramme de l'apprenant, la composition « possède » entre les classes « Crayon » et « Personne » est dédoublée et les doublons sont transférés vers les classes filles de « Crayon ». Ces différences se traduisent par une différence composée (B), appelée « dédoublement et transfert » dans la taxonomie TDP, à laquelle est associée la rétroaction suivante :

*Indiquer* : Tu dis que 'Stylo possède Corps' et que 'Feutre possède Corps'.
*Questionner* : Les relations 'Stylo-Corps' et 'Feutre-Corps' représentent-elles la même relation ?
*Proposer* : Tu dois les regrouper en une seule relation, en utilisant les classes Crayon et Corps.

L'apprenant a remplacé l'agrégation « possède » entre « Bouchon » et « Feutre » par une composition. De plus, l'orientation de la relation est inversée. Ces deux différences structurelles correspondent à une différence pédagogique composée (C), formée d'une erreur de type et d'une inversion de sens. La rétroaction associée est focalisée sur la différence d'inversion du sens de relation.

*Indiquer* : Tu dis que 'Bouchon possède Feutre'.
*Questionner* : Est-ce que 'Bouchon possède Feutre' ?
*Proposer* : Je dirais plutôt que 'Feutre possède Bouchon'.

Enfin, l'apprenant a inversé le sens de l'héritage entre « Feutre effaceur » et « Feutre ». Cette différence structurelle correspond à une différence pédagogique simple (D). Cette différence conduit à une rétroaction isolée, selon trois modalités :

*Indiquer* : Tu dis que 'Feutre est une sorte de Feutre effaceur'.
*Questionner* : Est-ce que 'Feutre est une sorte de Feutre effaceur' ?
*Proposer* : Je dirais plutôt que 'Feutre effaceur est une sorte de Feutre'.

Au total, quatre rétroactions sont produites pour ce diagramme, dont trois concernent des différences composées et une correspond à une différence simple. Chaque rétroaction peut se décliner en deux ou trois modalités, selon le cas, et donner lieu à des messages plus ou moins directifs. Ces messages sont présentés à l'apprenant de manière graduée, lors d'une étape spécifique dans Diagram, à l'issue de l'étape de relecture.

La figure 57 illustre un exemple d'affichage des rétroactions dans Diagram à l'issue de la relecture par l'apprenant. Les rétroactions (nommées « Remarques » dans l'interface) sont listées dans le volet droit de Diagram. Le diagramme de l'apprenant est différent de celui de l'exemple présenté précédemment mais il correspond à un diagramme réellement construit par un apprenant dans Diagram pour ce même exercice. La rétroaction présentée

concerne la représentation de la classe abstraite « Crayon » de la solution de référence sous forme d'une classe nommée « Concept ». Les trois messages de rétroaction focalisent la réflexion de l'apprenant sur l'abstraction de cette classe.

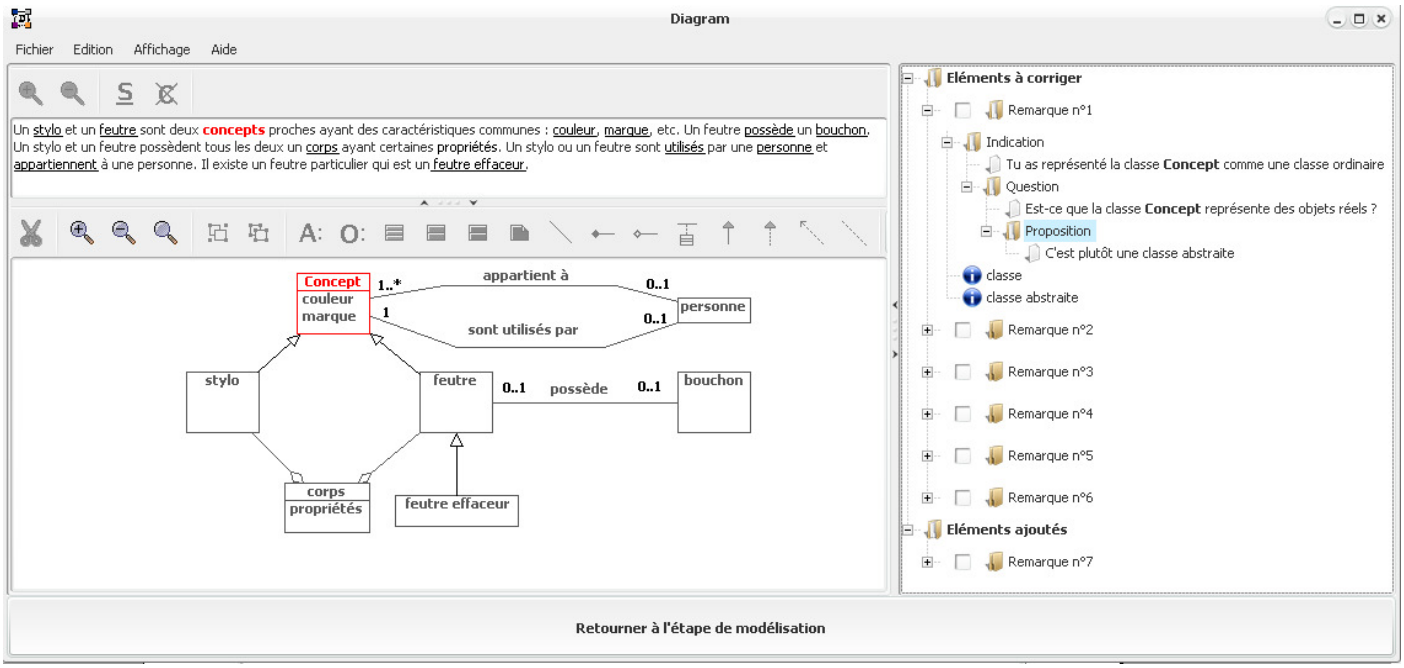



**Figure 57 : Interface graphique de l'étape d'affichage des rétroactions pédagogiques de Diagram**

# CHAPITRE 7

# L'implantation et l'évaluation du système ACDC

# PLAN DU CHAPITRE

Nous avons montré comment notre méthode d'appariement est appliquée pour l'analyse des diagrammes de classes UML construits par l'apprenant dans l'environnement Diagram et comment ses résultats sont utilisés pour fournir des rétroactions pédagogiques à l'apprenant en cours d'activité de modélisation. Nous allons maintenant détailler l'implantation effective, l'agencement des différentes fonctionnalités de notre module dans Diagram et l'évaluation de la qualité des résultats produits.

La méthode d'appariement est instanciée dans Diagram sous forme d'un composant logiciel programmé en JAVA, nommé ACDC (***A**utomatic **C**lass **D**iagrams **C**omparator*). Initialement, ce composant a été conçu et développé indépendamment de Diagram pour des besoins de standardisation du formalisme des diagrammes à comparer, de validation syntaxique, d'expérimentation et de réutilisation du composant. Enfin, Diagram était en cours de développement lors de la définition d'ACDC. Nous avons choisi une représentation des diagrammes fidèle au métamodèle d'UML, reposant sur le plugin uml2 du projet Eclipse. ACDC est ainsi capable de mesurer les similarités (et les différences) et d'apparier des diagrammes sauvegardés au format XMI (standard actuel utilisé dans de nombreux éditeurs UML) et de renvoyer ses résultats (listes d'erreurs syntaxiques, de motifs appariés et de différences structurelles identifiés) à la fois sous forme d'objets et de traces textuelles. ACDC a été intégré dans Diagram et deux modules ont été définis pour permettre de le lier avec le reste de l'environnement (les modules *Interface* et *Interaction* de Diagram) : un dispositif de conversion des modèles graphiques au format uml2 et un dispositif de conversion des résultats d'ACDC en différences pédagogiques.

L'évaluation de la qualité des systèmes d'appariement nécessite en général d'évaluer leur complexité, leur temps de calcul et de comparer les résultats du système aux résultats attendus (déterminés manuellement par un expert). Les résultats de cette comparaison sont ensuite quantifiés et qualifiés à l'aide de mesures qui expriment au minimum le pourcentage d'erreurs commises par le système. La pertinence de notre système peut être évaluée en fonction de sa robustesse et de ses performances effectives sur des diagrammes produits par l'apprenant. Le système doit être notamment capable de fournir des résultats rapidement (pour des besoins de rétroactions pédagogiques synchrones) et avec le moins d'erreurs possibles (pour que les rétroactions soient pertinentes et adaptées à l'apprenant). Nous avons ici la possibilité d'évaluer le diagnostic sur des diagrammes fictifs ou construits par des apprenants, que ce soit en cours d'activité de modélisation dans Diagram ou en dehors de l'environnement. L'évaluation de la pertinence des résultats doit prendre en compte à la fois les appariements et les différences dans ACDC. Nous avons réalisé la première évaluation du système ACDC pendant le premier semestre 2008, en dehors de l'environnement Diagram, sur un corpus de diagrammes UML préalablement construit par des apprenants.

## 7.1 Implantation du système dans Diagram

L'environnement Diagram est programmé en JAVA 1.5 et peut ainsi être exécuté sous différents types de plateformes où une machine virtuelle JAVA de cette version (ou ultérieure) est installée. Diagram repose sur une architecture MVC (***M**odel-**V**iew-**C**ontroller*) définie pour permettre la représentation et l'édition de modèles sémantiques quels qu'ils soient. Graphiquement, les modèles sont représentés et éditables via le composant générique de représentation de graphes JGraph [JGraph]. Les éléments manipulables dans les modèles sont définis dans un fichier XML conforme à une DTD (***D**ocument **T**ype **D**efinition*) décrivant leurs propriétés (forme, couleur, taille...), la manière

dont ils peuvent être utilisés (éléments et informations contenus ou liés). Cette représentation n'apporte pas d'information au niveau de la syntaxe et des spécificités propres à UML car elle n'est pas dédiée spécifiquement à la création de diagrammes UML.

Les activités de modélisation proposées à l'apprenant dans Diagram sont chargées et sauvegardées dans une archive zip. Cette archive contient l'énoncé à modéliser, le scénario (cf. partie 2.4), le modèle idéal correspondant à l'énoncé (et les liens textuels avec les expressions utilisés dans l'énoncé pour construire les éléments graphiques), l'état d'avancement du diagramme à charger dans l'interface et enfin des contraintes spécifiques au paramétrage de l'application pour l'exercice donné. Diagram trace au fur et à mesure dans des fichiers XML l'ensemble des actions réalisées par l'apprenant lors de l'activité de modélisation et enregistre l'état d'avancement du diagramme. Les diagrammes peuvent être exportés également sous forme d'images (jpg, png et bmp).

Notre système de diagnostic des diagrammes construit par l'apprenant joue le rôle d'interface entre les modules *Interface* et *Interaction* de Diagram pour la production de rétroactions pédagogiques. Il analyse le diagramme construit par l'apprenant à l'interface après sa conversion dans un format conforme au métamodèle d'UML et présente les résultats de cette analyse au module *Interaction* (les erreurs de cohérence, de syntaxe et les différences structurelles relevées sont converties en différences pédagogiques).

### 7.1.1 Utilisation du composant UML2 du projet EMP

Le système ACDC permettant de réaliser le diagnostic du diagramme construit par l'apprenant est développé sous forme d'un composant logiciel capable de comparer et d'apparier des diagrammes de classes conformes au langage UML 2.x. Ce langage est standardisé, c'est pourquoi il faut que les diagrammes manipulés soient conformes au métamodèle d'UML afin de les analyser correctement et rapidement. Nous n'avons pas choisi de conserver la représentation des modèles graphiques définie dans Diagram car elle n'est pas conforme au métamodèle d'UML. De plus, elle n'est pas adaptée à nos besoins d'appariement car les informations sur les propriétés graphiques y sont superflues, les liens sémantiques propres à UML n'y sont pas définis et enfin il est difficile d'interroger les éléments du diagramme pour connaître leur vision du reste du Diagram dans cette représentation. Nous n'avons pas non plus défini une instanciation du métamodèle d'UML étant donné que ce travail a déjà été réalisé dans de nombreux projets ou éditeurs liés à UML. Des instanciations complètes peuvent répondre à nos attentes notamment dans les éditeurs couramment utilisés mais leurs méthodes et codes sources ne sont que très rarement accessibles et utilisables librement. Nous nous sommes donc tournés vers une solution *open-source* issue des travaux de la communauté Eclipse et de leur plateforme de développement éponyme.

Le projet EMP (***E**clipse **M**odeling **P**roject*) [EMP] de la communauté Eclipse [Eclipse] fournit un ensemble unifié de cadres, d'outils, et d'implantation de normes liés à la modélisation. Le composant logiciel UML2 du sous-projet MDT (***M**odel **D**evelopment **T**ools*) [MDT] a attiré notre attention car il fournit une implantation fidèle du métamodèle actuel UML 2.1 [OMG UML] permettant de supporter le développement d'outils de modélisation, un encodage échangeable XML des modèles sémantiques en utilisant le standard XMI 2.1 [OMG XMI] recommandé par l'OMG et enfin des règles de validation des modèles très proches des règles de cohérence de [Seuma Vidal et *al.* 2005]. Nous avons testé les possibilités fournies par le composant UML2 et nous sommes appropriés son utilisation avant de le

retenir pour représenter le métamodèle d'UML, les diagrammes de classes UML et pour valider syntaxiquement ces derniers dans ACDC. UML2 permet de créer et de manipuler simplement un diagramme de classes en interne directement par des appels de méthodes écrites en JAVA. Le métamodèle d'UML implanté est complet et correspond aux documents de description d'UML 2.1 fournis par l'OMG.

Le composant ACDC est capable de comparer et d'apparier des diagrammes stockés au format XMI, qu'ils proviennent de Diagram en étant créés à l'interface et ensuite convertis (modèle graphique vers modèle XMI) ou qu'ils proviennent d'autres éditeurs UML[29] permettant d'exporter des diagrammes au format XMI (sans profil UML spécifique). Les diagrammes sont sauvegardés dans des fichiers portant l'extension *.uml2* et peuvent être visualisés sous forme arborescente directement dans l'éditeur de fichier XMI d'Eclipse (cf. annexe 4).

### 7.1.2 Conversion du modèle graphique en modèle uml2

Étant donné que les modèles graphiques et les modèles utilisés dans ACDC n'ont pas le même formalisme, il est nécessaire de mettre en place un sous-système de conversion de diagrammes entre ces deux formats qui permet de faire la liaison entre le module *Interface* et le module de *Diagnostic*.

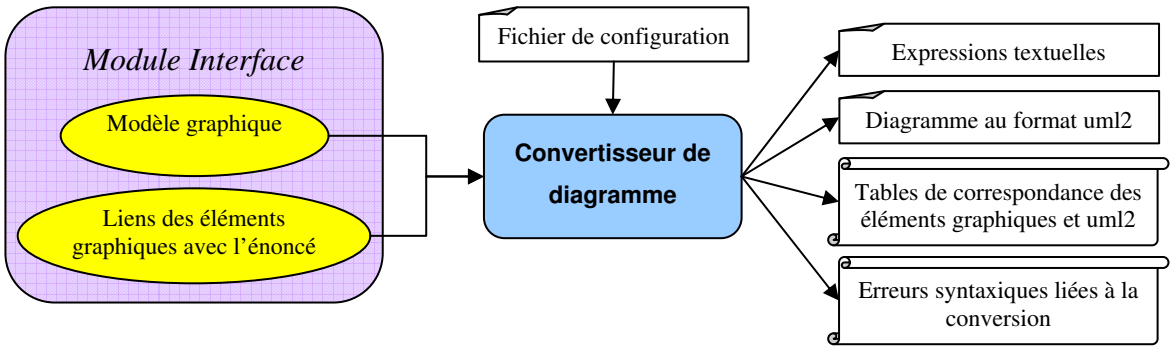



**Figure 58 : Convertisseur de modèle graphique implanté dans Diagram**

Le **convertisseur de modèles** implanté permet de transformer le modèle graphique en un diagramme conforme au métamodèle UML 2.x pouvant être traité directement par le système de comparaison et d'appariement. Le diagramme converti à partir du modèle graphique et ses éléments sont nommés respectivement par la suite par les termes **diagramme uml2** et **éléments uml2** correspondant à l'extension du fichier XMI créé par le composant UML2 après sa conversion. Pour pouvoir transformer les différents éléments graphiques en éléments uml2, nous avons étudié la DTD décrivant le cadre dans lequel les modèles graphiques doivent être exprimés dans Diagram, le fichier XML conforme à cette DTD décrivant les propriétés des diagrammes de classes et enfin le métamodèle UML 2.x. Trois types d'éléments graphiques sont définis dans Diagram : les **éléments unaires** (les sommets des modèles), les **éléments binaires** (les arcs des modèles), les **éléments communs** des sommets et des arcs. Nous présentons dans le tableau de la figure 59 la correspondance entre les éléments graphiques et les éléments uml2 :

[29] Nous avons réalisé des tests avec l'éditeur MagicDraw UML pendant l'expérimentation de notre système mais un grand nombre d'autres éditeurs dans leur version complète (Rational, Together, Poseidon...) permettent l'exportation des diagrammes UML dans le format XMI sans utiliser de profil UML spécifique. De plus, le composant UML2 permet de convertir des diagrammes au format XMI 1.x dans le format XMI 2.x s'il respecte en tout point la norme.

| Éléments graphiques de Diagram | | Éléments uml2 |
|---|---|---|
| Elément graphique unaire : *UnaryGraphicCellElement* | Type = 'classe' | Classe : *Class* avec l'attribut *abstract* = 'false' |
| | Type = 'classe abstraite' | Classe : *Class* avec l'attribut *abstract* = 'true' |
| | Type = 'classe interface' | Interface : *Interface* |
| | Type = 'note' | Commentaire : *Comment* |
| Elément graphique binaire : *BinaryGraphicCellElement* | Type = 'lien d'héritage' | Relation d'héritage : *Generalization* |
| | Type = 'association' ou 'agrégation' ou 'composition' | Relation d'association : *Association* avec le type d'agrégation défini au niveau des fins d'association |
| | Type = 'lien de dépendance' ou 'lien de réalisation' | Relation de dépendance : *Dependency* |
| | Type = 'lien d'attachement' | Pas de correspondance directe |
| | Type = 'lien d'une classe-association' | Pas de correspondance directe |
| Elément graphique commun : *CommonPartGraphicElement* | Type = 'attribut' | Attribut : *Property* |
| | Type = 'opération' | Opération : *Operation* |

**Figure 59 : Correspondance des éléments graphiques un à un avec les éléments uml2**

Des éléments graphiques tels que les liens d'attachement et les liens de classe d'association n'ont pas d'équivalents directs dans le formalisme actuel d'UML2 car ils sont définis implicitement et renvoient à d'autres éléments, respectivement les commentaires et les classes d'association. Au contraire, les fins d'association encapsulent les rôles, les multiplicités et le type d'agrégation d'une relation d'association dans le métamodèle d'UML. Elles n'ont pas d'équivalent sous forme d'éléments dans le modèle graphique (ce sont seulement des informations encapsulées directement dans l'élément graphique binaire correspondant à une association). Les éléments uml2 de type 'classe d'association', 'commentaire' et 'fin d'association' sont mis en correspondance avec plusieurs informations ou éléments graphiques présentés dans le tableau suivant.

| Éléments uml2 | Éléments graphiques de Diagram |
|---|---|
| Classe d'association : *AssociationClass* | Trois éléments graphiques :<br>Un élément unaire de type 'classe' ou 'classe abstraite' lié par un élément binaire de type 'lien de classe-association' à un élément binaire de type 'association' |
| Fin d'association : *Property* | Pas de correspondance :<br>Une transposition est possible à partir des informations sur les rôles et les multiplicités directement contenues dans les éléments binaires de type 'association' ou 'agrégation' ou 'composition' |
| Commentaire : *Comment* | Deux éléments graphiques :<br>Un élément unaire de type 'note' lié par un élément binaire de type 'lien d'attachement' à un autre élément graphique |

**Figure 60 : Éléments uml2 n'ayant pas directement d'équivalent avec les éléments graphiques**

Lors de la conversion, les éléments graphiques sont parcourus et convertis un à un en éléments UML par des **fonctions de transformation**. Ces fonctions extraient les données des éléments graphiques nécessaires à la création des éléments uml2 et appellent directement les fonctions de création des éléments définies dans le composant UML2

d'Eclipse. Les éléments graphiques unaires sont parcourus et convertis en premier puis ce sont les éléments graphiques binaires qui sont convertis car ils relient les éléments unaires (les éléments uml2 équivalents doivent préexister pour être liés à des relations). Si un élément unaire ou binaire contient des éléments graphiques communs alors ils sont créés à la suite de l'élément les contenant et y sont encapsulés. La création des fins d'association est réalisée au moment de la création de l'association qui les contient. La création des classes d'association est réalisée juste après la création des éléments unaires car elles sont à la fois des sommets et des arcs. Plusieurs tables mettent en correspondance les éléments graphiques et uml2 pour connaître directement l'élément graphique correspondant à un élément uml2 et inversement. Si un élément graphique ne peut être converti en élément UML, une erreur syntaxique est créée et l'élément graphique n'aura pas d'équivalent dans les tables d'association des éléments. L'erreur créée encapsule un code d'erreur, un message d'erreur la décrivant, l'élément graphique associé à l'erreur (pour pouvoir le repérer dans le diagramme) et enfin l'élément uml2 s'il est possible d'en créer un moyennant la correction de l'erreur :

*syntaxError = (errorCode, errorMessage, graphicalElement, uml2Element)*

Dans Diagram, les éléments graphiques sont liés aux expressions ayant permis de les créer. Lors de la conversion, un fichier de propriétés est associé à chaque diagramme pour recenser les expressions de l'énoncé ayant permis de créer les différents éléments des diagrammes ; ces expressions sont prises en compte notamment lors de l'évaluation des similarités des noms des éléments uml2 (cf. partie 5.4.5) car les utilisateurs de Diagram ont la possibilité de les changer au cours de l'activité de modélisation.

### 7.1.3 Architecture et fonctionnement d'ACDC dans Diagram

L'ensemble du système ACDC et des modules de conversion sont paramétrés à l'aide d'un fichier de propriétés listant ses options de configuration. Ce fichier est chargé au démarrage de l'application pour initialiser tous les paramètres du système. Ce fichier de configuration, reporté en annexe 9, liste les options relatives :

- à la conversion d'un diagramme JGraph dans le formalisme du plugin uml2 ;
- aux types et à la manière d'extraire les motifs d'un modèle ;
- aux poids des différents critères pris en compte dans les calculs de similarité ;
- à la propagation, la dérivation et la restructuration ;
- aux différences spécifiques ou générales applicables aux motifs ;
- aux choix de correspondances univoques et/ou multivoques ;
- aux paramètres des stratégies appliquées lors du choix de l'appariement final ;
- à la prise en compte ou non de la validation syntaxique ;
- à la langue utilisée (anglais ou français) ;
- au traçage du comportement du système (mode de débogage).

Le diagnostic du diagramme de l'apprenant est effectué une fois que le diagramme de l'apprenant est converti et sauvegardé dans un fichier au format uml2. Sa cohérence et sa syntaxe sont analysées par un **vérificateur de cohérence** reposant sur celui défini dans le composant UML2 d'Eclipse. Les erreurs de syntaxe repérées lors de la conversion sont fusionnées avec celles repérées par le vérificateur de cohérence.

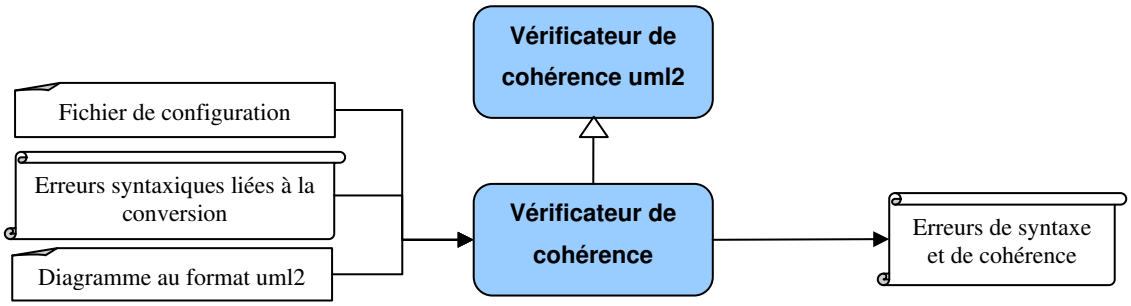



**Figure 61 : Vérificateur de cohérence**

Parallèlement à la vérification de la cohérence du diagramme de l'apprenant, ce dernier est comparé et apparié au diagramme de référence défini par un expert pour le même énoncé. Ce diagramme de référence est exprimé également dans le format uml2. Il est stocké dans l'archive zip associé à l'exercice que l'apprenant a à réaliser et qui est chargé dans Diagram. Il peut avoir été construit soit directement dans Diagram par l'expert et converti en fichier uml2, soit dans autre éditeur permettant d'exporter le diagramme construit dans le format .uml2. Les deux diagrammes et les expressions textuelles liées à l'énoncé (si elles existent) sont fournis en entrée d'ACDC dont l'architecture est reportée dans la figure 62.

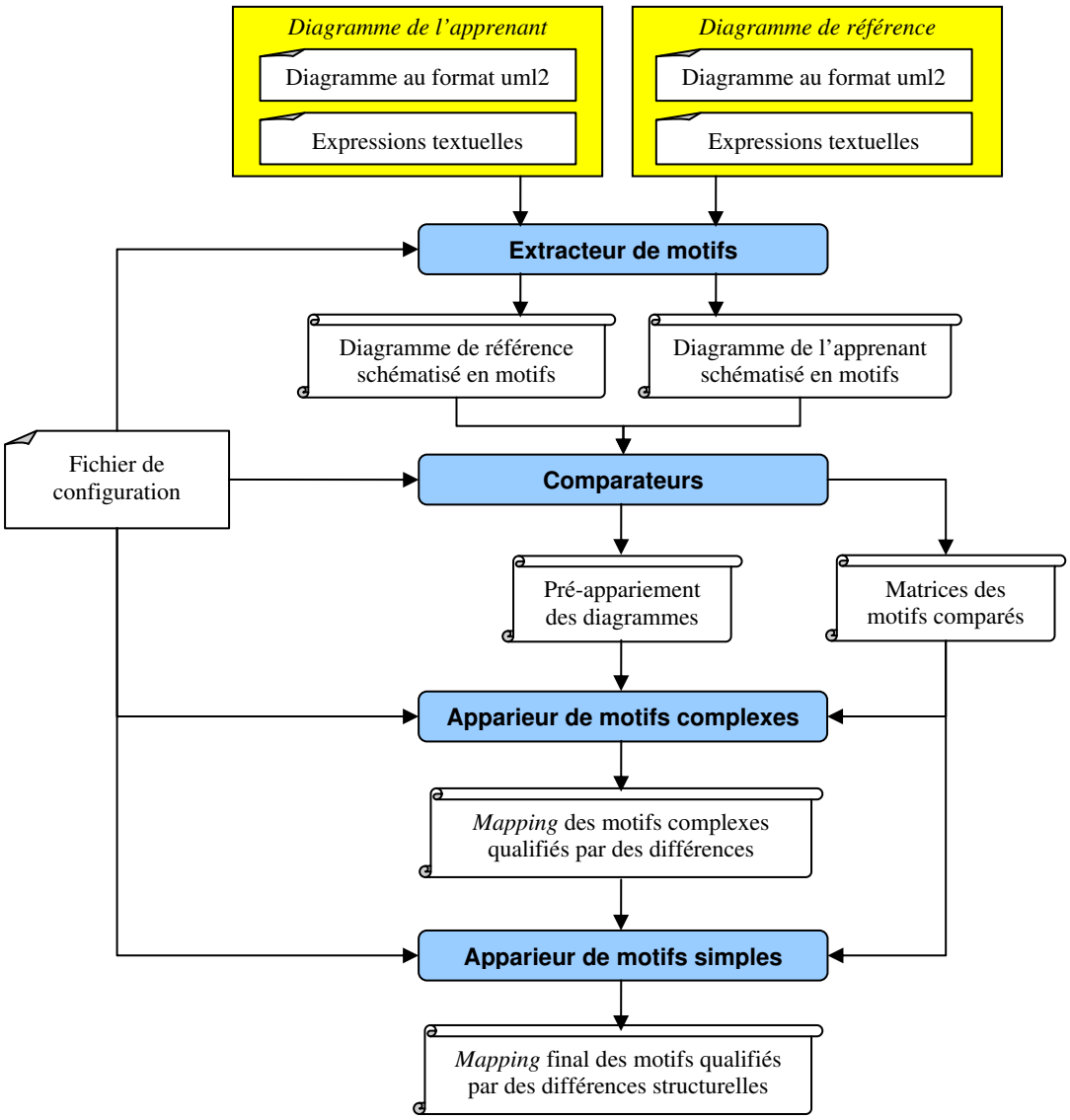



**Figure 62 : Architecture d'ACDC**

Les deux diagrammes en entrée du module ACDC sont schématisés en motifs simples et complexes. Chaque élément UML est encapsulé dans un motif simple (cf. partie 5.2). Les motifs complexes encapsulent ensuite les motifs simples. Les options de schématisation des diagrammes sont définies dans le fichier de configuration permettant de paramétrer ACDC. Nous avons défini plusieurs options de schématisation : la première permet de schématiser les motifs complexes sous forme de chemins. Par exemple, chaque feuille d'un arbre d'héritages allant jusqu'à la racine et chaque extrémité d'une chaîne d'associations peuvent être liées par différents chemins. Cette structuration permet de comparer tous les chemins possibles entre les différents motifs simples des diagrammes et ainsi d'avoir une vision très précise des déplacements de motifs. Néanmoins, elle présente les inconvénients de demander beaucoup de comparaisons et de traitements pour pouvoir résoudre certains conflits lors des appariements locaux et du choix de l'appariement final (à cause notamment des visions locales de l'appariement dans chacun des chemins). La seconde option de schématisation est celle que nous avons présentée dans la partie 5.2 : les diagrammes y sont schématisés sous forme de chaînes d'associations et d'arbres d'héritages. Moins de motifs complexes sont identifiés et moins de traitements sont effectués lors de l'évaluation de leurs similarités (et différences) et donc de leur appariement.

Une fois que les deux diagrammes sont schématisés en motifs simples et complexes, ils sont confrontés par les comparateurs dont le comportement est décrit de manière générale dans la partie 5.4.4 et plus précisément dans l'annexe 5. Les critères pris en compte dans les comparateurs sont définis dans le fichier de configuration d'ACDC. En sortie, un pré-appariement est obtenu à l'échelle des deux diagrammes comparés et localement à chaque couple de motifs comparés. Les résultats des comparateurs sont factorisés et agencés sous forme de matrices de motifs comparés et triés par score de similarité. Chaque ligne d'une matrice correspond à un couple de motifs comparés auquel est associé un score de similarité et des différences structurelles locales qui ont été définies par les comparateurs. L'apparieur de motifs complexes puis l'apparieur de motifs simples se chargent ensuite de sélectionner les appariements de motifs à ajouter dans le *mapping* final des diagrammes. Les appariements de motifs sont ajoutés au fur et à mesure dans le *mapping* en fonction de leur score de similarité et de leur cohérence avec le *mapping* courant (cf. partie 5.5). En sortie, l'apparieurs de motifs constitue un *mapping* des motifs appariés qualifiés par des différences structurelles. Une liste de différences structurelles peut être facilement extraite en parcourant les appariements du *mapping*.

Les différences pédagogiques sont élaborées à partir des erreurs de syntaxe relevées par le vérificateur de cohérence, le convertisseur sur le diagramme de l'apprenant et enfin des différences structurelles où interviennent les motifs simples (les appariements des motifs complexes ne sont pas utilisés lors de la production de rétroactions pédagogiques). Le **convertisseur de différences** (cf. figure 63) se charge de transformer ces différences dites structurelles en différences pédagogiques définies dans la TDP. Les données apparaissant dans les différences structurelles sont filtrées, ré-agencées et converties en différences pédagogiques par des fonctions de transformations dédiées à chaque type de différences pédagogiques. Le système utilise les tables de correspondance des éléments graphiques et uml2 pour pouvoir associer à chaque différence pédagogique, le ou les éléments graphiques y intervenant dans le diagramme de l'apprenant. Cette information est utilisée ensuite par le module *Interaction* qui a besoin de connaître quels sont les éléments graphiques entrant en jeu dans une différence pédagogique notamment pour pouvoir interagir directement à l'interface en faisant ressortir par exemple en couleur un élément du diagramme de l'apprenant.

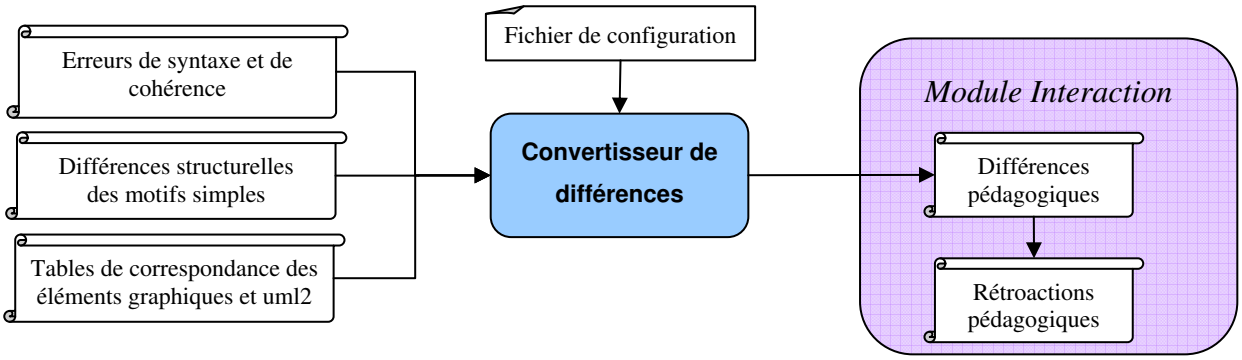



**Figure 63 : Convertisseur de différences**

## 7.2 Évaluations de la qualité du diagnostic

La qualité des résultats produits et la vitesse de production sont des facteurs primordiaux pour fournir des rétroactions pédagogiques pertinentes à l'apprenant. Nous avons évalué le système ACDC indépendamment de l'environnement Diagram au niveau de ses performances de calcul (rapidité de calcul) et de la qualité des différences et des appariements produits par rapport à ceux attendus. Pour évaluer ces deux facteurs, nous avons collecté des diagrammes de classes UML construits par les apprenants. À partir de ce corpus de diagrammes, nous avons choisi trois exercices pour calibrer et valider le système ACDC. Nous utilisons quatre mesures standards pour mesurer la qualité des résultats.

### *7.2.1 Corpus d'exercices et de diagrammes à disposition*

L'utilisabilité, l'interaction et les fonctionnalités pédagogiques de Diagram telles que les reformulations des éléments modélisés par l'apprenant (sans le module de diagnostic) ont été testées ces dernières années pendant onze sessions, sur dix-sept exercices différents. Chaque session s'est déroulée lors d'une séance de TP (**T**ravaux **P**ratiques) de trois heures sur un public d'étudiants de deuxième année de DEUST ISR (**D**iplôme d'**É**tudes **U**niversitaires **S**cientifiques et **T**echniques en **I**nformatique, **S**ystèmes et **R**éseaux). Ces étudiants sont des novices qui n'ont pas eu au préalable d'enseignement de la programmation orientée objet. Ces séances de TP ont pour objectif de mettre en application les connaissances et les concepts de base de la modélisation orientée objet de niveau analyse qu'ils ont pu aborder de manière théorique en cours et appliquer sur papier en TD (**T**ravaux **D**irigés). Lors de ces séances, les étudiants (répartis en binôme ou individuellement) ont été amenés à construire des diagrammes de classes UML de niveau analyse à partir d'un énoncé textuel décrivant les spécifications à représenter. Le nombre d'exercices de construction de diagrammes de classes UML à réaliser varie dans les différentes séances (de un à cinq). L'enseignant du module de modélisation orientée objet est présent lors des séances. Il répond aux questions des étudiants et leur prodigue des conseils, des remarques mais ne donne la solution d'un exercice qu'au cours du TD suivant le TP. La moitié des étudiants ont utilisé Diagram et l'autre moitié un éditeur classique, Objecteering.

Les diagrammes de classes UML construits par les apprenant lors de ces séances ont été collectés pour être analysés ensuite. Nous avons ainsi à notre disposition un peu plus de trois cents diagrammes sous forme d'images, de fichiers de sauvegarde de l'éditeur Objecteering et également sous forme d'enregistrement vidéo pour les dernières sessions portant sur l'expérimentation des fonctionnalités pédagogiques de Diagram [Alonso 2009]

[Alonso *et al.* 2008]. Tous ne sont pas exploitables étant donné que pour certains exercices, les diagrammes produits peuvent se révéler très incomplets (certains étudiants ayant juste eu le temps de commencer l'exercice) ou en nombre insuffisants (les étudiants n'ont pas eu le temps de faire tous les exercices demandés lors d'une séance). C'est donc à partir d'un sous-ensemble représentatif de ce corpus de diagrammes de classes UML que nous avons paramétré et évalué hors-ligne le système ACDC.

### *7.2.2 Méthodologie*

Trois groupes de diagrammes de classes UML (quatre-vingt deux diagrammes en tout), correspondant à trois exercices de complexité croissante, ont été utilisés pour tester et valider la qualité ainsi que la robustesse d'ACDC. Nous avons retenu ces trois exercices car ils représentent un nombre de diagrammes suffisamment important (plus de vingt diagrammes) et des diagrammes suffisamment variés pour paramétrer et évaluer le système ACDC sur des cas différents. De plus, les solutions de référence, déterminées pour chacun des exercices par l'enseignant de DEUST ISR, incluent chacune des spécificités et des difficultés particulières. Nous reportons dans le tableau suivant le nombre de chacun des motifs apparaissant dans les diagrammes de référence :

| Nombres de | Exercice 1 | Exercice 2 | Exercice 3 |
|---|---|---|---|
| **Motifs complexes** | **3** | **3** | **5** |
| **Motifs simples** | **24** | **51** | **44** |
| Classificateurs | 7 | 9 | 14 |
| Relations | 7 | 10 | 14 |
| Caractéristiques | 2 | 22 | 10 |
| Fins d'association | 8 | 10 | 6 |

**Figure 64 : Répartition des éléments dans les diagrammes de référence**

Nous présentons dans les trois sous-parties suivantes les énoncés, les spécificités et les diagrammes de référence de chacun des trois exercices retenus pour l'évaluation de l'ACDC.

#### 7.2.2.1 Premier exercice

Le premier exercice utilisé pour l'évaluation est l'exercice « Stylo et Feutre » présenté dans la partie 1.4.1. Nous reportons dans la figure 65 l'énoncé ainsi que le diagramme de référence proposé par l'enseignant. Le diagramme de référence pour cet exercice contient un petit arbre d'héritages (quatre classes réparties sur trois niveaux) et peu d'éléments (vingt quatre en tout). Il présente néanmoins deux spécificités : une classe implicite qui n'est pas exprimée dans l'énoncé (la classe abstraite « Crayon ») et un couple de classes reliées par deux relations représentant deux rôles distincts (deux associations nommées respectivement « appartient à » et « est utilisé par »). La classe abstraite « Crayon » factorise les propriétés et les comportements communs de ses classes filles (attributs et relations) ; ce choix de modélisation évite ainsi de représenter de manière redondante certaines informations.

Un stylo et un feutre sont deux concepts proches ayant des caractéristiques communes : couleur, marque, etc. Un feutre possède un bouchon. Un stylo et un feutre possèdent tous les deux un corps ayant certaines propriétés. Un stylo ou un feutre sont utilisés par une personne et appartiennent à une personne. Il existe un feutre particulier qui est un feutre effaceur.

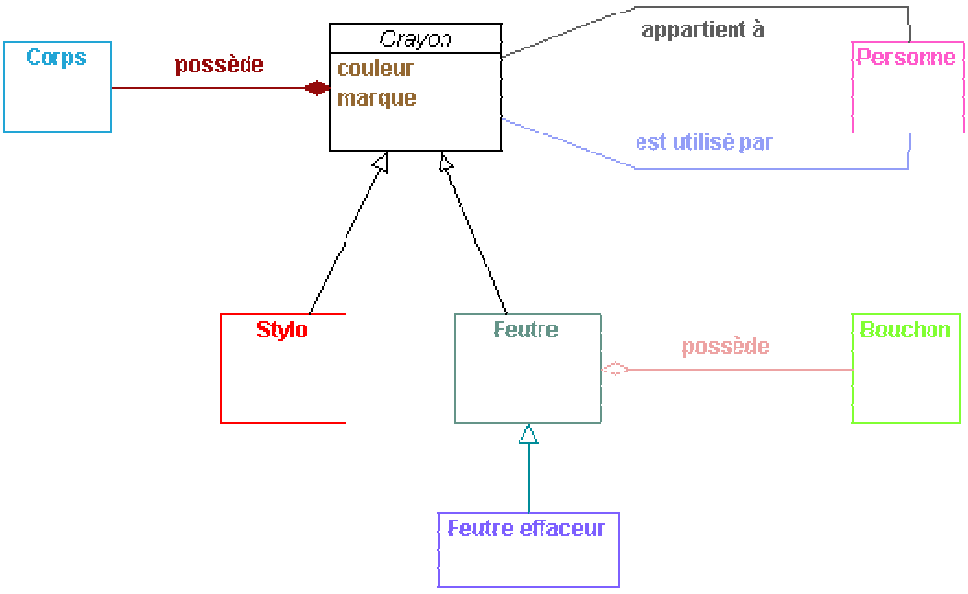



**Figure 65 : Enoncé et diagramme de référence de l'exercice « Stylo et Feutre » (avec Diagram)**

### 7.2.2.2 Second exercice

L'énoncé et le diagramme de référence (construit à l'aide de l'éditeur MagicDraw UML) du second exercice nommé « Cirque » sont reportés dans la figure 66. Nous avons utilisé ce diagramme dans l'exemple de production des différences structurelles par l'ACDC dans la partie 5.6. Ce diagramme de référence contient plus d'éléments que celui du premier exercice. Il inclut deux arbres d'héritages distincts (à deux niveaux d'abstraction) reliés entre eux par plusieurs relations d'association (formant une chaîne d'associations). Des attributs sont définis en plus grand nombre également dans chacune des classes à représenter dans le diagramme. Les spécificités de cet exercice résident dans son énoncé car il décrit des instances de classes et définit certains attributs de classes avec plusieurs expressions relatives à un même concept (la capacité et la taille du camion par exemple). Il ne faut donc pas représenter les objets sous forme de classes et les attributs redondants.

> Les membres du personnel du cirque sont caractérisés par un numéro (en général leur numéro INSEE), leur nom, leur prénom, leur date de naissance et leur salaire. On souhaite de surcroît stocker les pseudonymes des artistes et le numéro du permis de conduire des chauffeurs de poids lourds.
>
> Les artistes sont susceptibles d'assurer plusieurs numéros, chaque numéro étant caractérisé par un code, son nom, le nombre d'artistes présents sur scène et sa durée. De plus, on souhaite savoir l'instrument utilisé pour les numéros musicaux, l'animal concerné par les numéros de dressage et le type des acrobaties (contorsionnisme, équilibrisme, trapèze volant…).
>
> Par ailleurs, chaque numéro peut nécessiter un certain nombre d'accessoires caractérisés par un numéro de série, une désignation, une couleur et un volume. On souhaite également savoir, individuellement, quels artistes utilisent quels accessoires.
>
> Enfin, les accessoires sont rangés après chaque spectacle dans des camions caractérisés par leur numéro d'immatriculation, leur marque, leur modèle et leur capacité (en volume). Selon la taille du camion, une équipe plus ou moins nombreuse de chauffeurs lui est assigné (de un à trois chauffeurs).

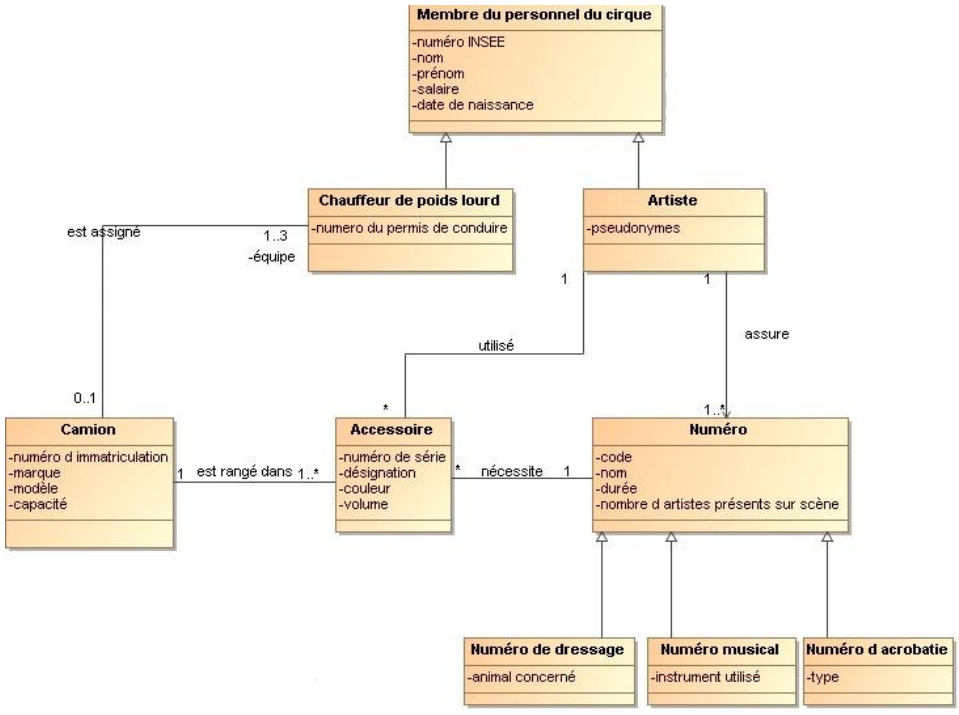



**Figure 66 : Enoncé et diagramme de référence de l'exercice « Cirque » (avec MagicDraw UML)**

### 7.2.2.3 Troisième exercice

Le dernier exercice nommé « Carte Géographique » est illustré dans la figure 67. Le dernier diagramme de référence ne contient qu'un arbre d'héritages (à quatre niveaux). Deux classes définies dans le diagramme de référence sont implicites dans l'énoncé (« Donnée simple » et « Donnée complexe »). Cet exercice demande une analyse plus approfondie pour définir la hiérarchie d'héritages centrale du diagramme de référence et pour modéliser correctement les relations de généralisation et de composition entre les différentes classes du diagramme. De nombreuses redondances sont présentes dans cet énoncé au niveau des attributs et des relations à représenter au bon endroit dans la hiérarchie d'héritages. Cet énoncé peut induire d'autres solutions alternatives.

Une carte géographique est caractérisée par une échelle, la longitude et la latitude de son coin inférieur gauche, la hauteur et la largeur de la zone couverte par la carte.

La carte comporte un ensemble de données géographiques de natures diverses.

Les villes et les montagnes sont repérées par un point unique. Chaque point a deux coordonnées x et y calculées par rapport au coin inférieur gauche de la carte.

Un nom est associé à chaque donnée géographique repérée par un point.

Les routes et les rivières sont repérées par des lignes brisées, c'est à dire par un ensemble de points correspondant aux extrémités de ses segments de droite. Les routes et les rivières ont des noms et des épaisseurs de traits.

Les lacs, mers et forêts sont représentés par des régions caractérisées par un nom et une couleur de remplissage. Une région est une ligne brisée refermée sur elle-même.

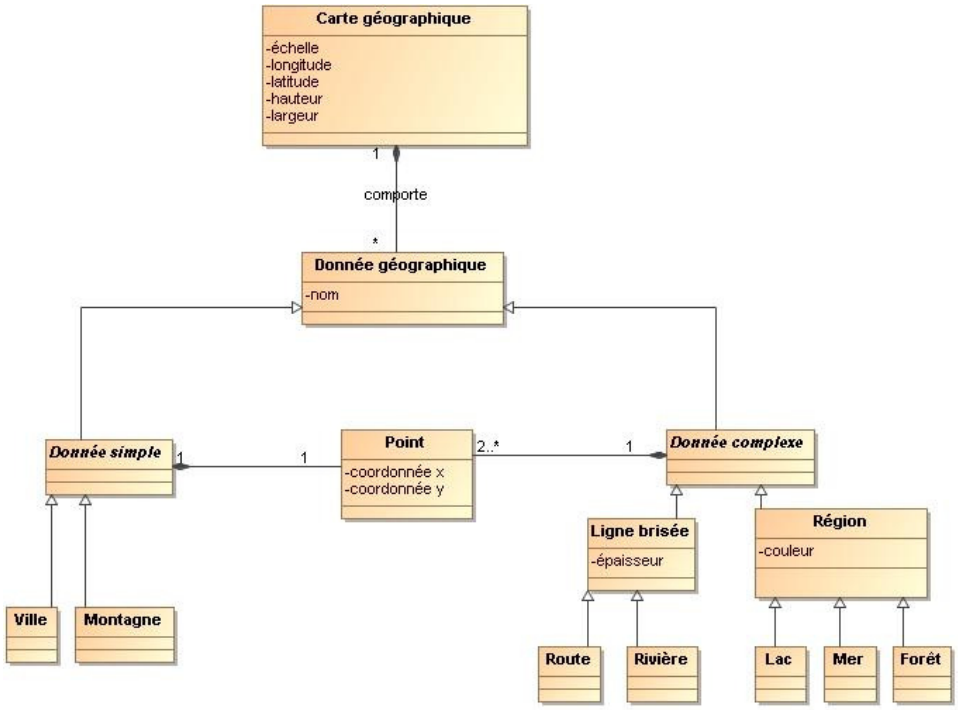



**Figure 67 : Enoncé et diagramme de référence de l'exercice « Carte géographique » (avec MagicDraw UML)**

#### 7.2.2.4 Transcription des diagrammes et méthodologie suivie

Nous avons transcrit dans un premier temps les diagrammes de classes UML utilisés pour l'évaluation au format uml2 (XMI 2.x) en les saisissant dans l'éditeur UML professionnel MagicDraw UML (le convertisseur de diagrammes n'étant pas encore en place à ce moment là dans Diagram) car il fournit la possibilité d'exporter des diagrammes conformes à cette norme et directement exploitables par notre système. Nous avons ensuite paramétré et testé dans un premier temps les critères intervenant dans le calcul des fonctions de similarité et le fonctionnement général d'ACDC à partir des diagrammes de classes UML construits par les apprenants pour le premier exercice. Ensuite, nous avons utilisé le système sur le second groupe de diagrammes de classes du deuxième exercice. Quelques incohérences ont été repérées et corrigées en veillant à ne pas dégrader la qualité d'appariement du premier jeu de test. Enfin, le troisième groupe a servi à valider le système sans aucune modification des critères et des algorithmes embarqués dans le système ACDC.

Pour pouvoir mesurer la qualité et la pertinence des appariements produits par le système, nous avons déterminé à la main les différences et les appariements que devrait identifier le système ACDC pour chacun des diagrammes des apprenants. Ensuite nous avons comparé les résultats que devrait trouver le système (appariements et différences réels ou attendus) par rapport à ceux qu'il trouve en réalité (appariements et différences du système). L'écart entre ces deux résultats est évalué à l'aide de quatre mesures standards de qualité.

### *7.2.3 Mesures de qualité utilisées*

Les mesures standards de qualité utilisées pour évaluer la pertinence des résultats produits en termes d'appariements sont les mêmes que celles utilisées actuellement pour évaluer les systèmes d'appariement de schémas et d'ontologies. Ces dernières années, la majorité des chercheurs se sont accordés au niveau de l'évaluation des systèmes d'appariement de schémas pour pouvoir comparer les différents systèmes d'appariement existants sur une base de critères et de mesures communes. Les mesures de qualité utilisées ont été introduites notamment dans [Melnik *et al.* 2002], [Do *et al.* 2002], [Do & Rahm 2007] et [Giunchiglia *et al.* 2007]. Nous avons retenu également ces mesures car elles permettent d'analyser les résultats par rapport aux erreurs et aux lacunes d'un système en les comparant à des résultats optimaux. Les **résultats de l'apparieur** (résultats de l'ACDC) sont comparés aux **appariements réels déterminés manuellement** (résultats réels). Les appariements peuvent être classés en quatre ensembles *A*, *B*, *C et D* :

- Les **vrais positifs (*true positives*) *B*** sont les appariements identifiés correctement par le système. Plus cet ensemble est important, plus les résultats produits par le système sont pertinents (corrects).
- Les **faux négatifs (*false negatives*) *A*** sont les appariements non identifiés par le système mais qui devraient l'être (ce sont des « oublis » du système).
- Les **faux positifs (*false positives*) *C*** sont les appariements faux ou mal étiquetés que propose le diagnostic automatique (ce sont des « erreurs » commises par le système).
- Les **vrais négatifs (*true negatives*) *D*** sont les appariements faux ou mal étiquetés que ne propose pas le système. Ce sont des « erreurs » que pourrait commettre le système mais qu'il n'a pas commises. Cet ensemble n'est pas pertinent dans notre cas et il est difficilement mesurable car il peut y avoir autant d'erreurs possibles

que d'appariements multivoques des éléments du diagramme de l'apprenant à ceux du diagramme de référence.

Le schéma suivant expose l'intersection des appariements attendus et des appariements identifiés en réalité par un système d'appariement. L'objectif est d'obtenir l'intersection (*B*) la plus grande possible entre les appariements attendus et ceux réellement identifiés, c'est-à-dire minimiser les « oublis » (*A*) et les « erreurs » (*C*).

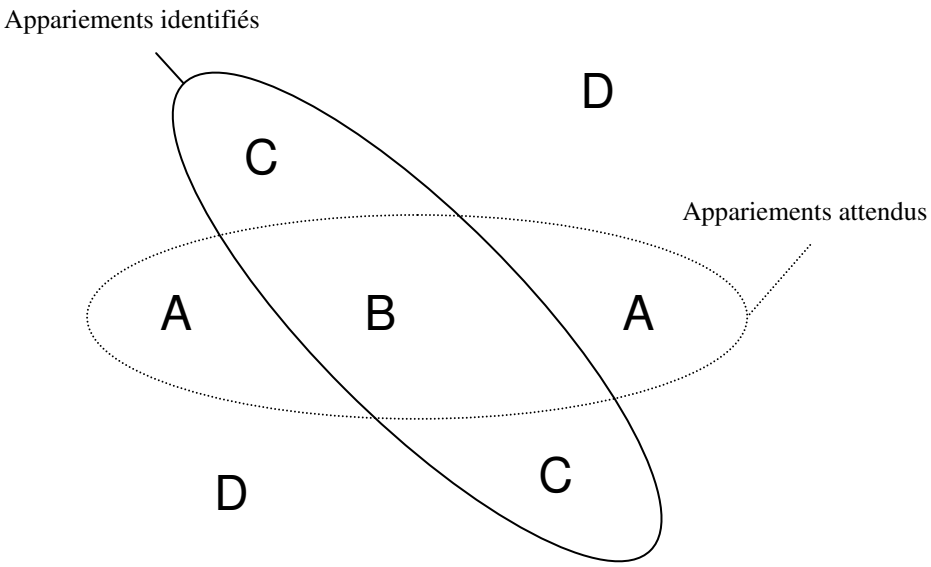



**Figure 68 : Relations entre les appariements attendus et ceux réellement identifiés par l'apparieur**

Les appariements sont classés dans les ensembles *A*, *B* et *C*. La qualité des résultats produits peut être évaluée ensuite suivant quatre mesures de qualité : la ***Précision*** (***Precision***), le ***Rappel*** (***Recall***), la ***F-Mesure*** (***F-Measure***) et l'***Overall***, qui sont basées sur les cardinalités des trois ensembles *A*, *B* et *C*. Ces mesures sont définies ainsi :

- *Précision* = $|B| / (|B| + |C|)$. La *Précision* mesure l'exactitude des résultats. Elle reflète la part d'appariements corrects trouvés par rapport à tous ceux identifiés (corrects ou erronés). La valeur calculée de la *Précision* varie dans un intervalle [0,1]. Plus la précision du système s'approche de un, moins il y a d'appariements erronés ou mal étiquetés identifiés par le système (par rapport à tous ceux relevés).

- *Rappel* = $|B| / (|B| + |A|)$. Le *Rappel* mesure la complétude des résultats produits. Il spécifie la part d'appariements corrects trouvés par rapport à tous ceux qui devraient être repérés (corrects et non identifiés). La valeur calculée du rappel varie dans un intervalle [0,1]. Plus le rappel du système tend vers un, moins il y a d'appariements « oubliés » par le système (par rapport à tous ceux qu'il devrait identifier).

- *F-Mesure* = $2$ (*Précision* * *Rappel*) / (*Précision* + *Rappel*) = $2 |B| / (|C| + 2 |B| + |A|)$. La *F-Mesure* est la moyenne harmonique des mesures de *Précision* et de *Rappel*. Elle permet d'avoir une vision plus synthétique des résultats du système en prenant à la fois en compte les appariements corrects, oubliés et erronés. La valeur calculée de la *F-Mesure* varie dans un intervalle [0,1]. Plus le résultat du calcul de la *F-Mesure* tend vers un, meilleurs sont les résultats de l'apparieur.

- *Overall = Rappel * (2 – 1 / Précision) = (|B| - |C|) / (|B| + |A|)*. Contrairement aux trois mesures précédentes, cette dernière a été conçue et utilisée spécialement dans le contexte de l'appariement de schémas [Melnik et al. 2002] [Do & Rahm 2002]. Tout comme la *F-Mesure*, l'*Overall* est une mesure combinée de la *Précision* et du *Rappel*. L'idée ici est de quantifier les efforts qui seraient nécessaires pour modifier l'appariement proposé par le système (à l'issue de l'appariement, *post-match*), c'est-à-dire en ajoutant les « oublis » (*A*) et en supprimant les « erreurs » (*C*) dans l'appariement proposé. La valeur calculée varie dans un intervalle [-1,1]. Plus le résultat du calcul de l'*Overall* tend vers un, meilleurs sont les résultats de l'apparieur. Si le nombre de *faux positifs* (*C*) dépasse le nombre de *vrais positifs* (B), c'est-à-dire si la *Précision* est inférieure à 0,5 alors la valeur obtenue par cette mesure est négative. La *F-Mesure* est plus optimiste que l'*Overall*.

### 7.2.4 Évaluations hors-ligne

L'application des quatre mesures de qualité sur les résultats de l'ensemble des trois exercices est reportée dans le tableau de la figure 69. Nous précisons dans l'annexe 10 sous forme d'histogrammes les résultats pour chacun des diagrammes étudiés dans cette évaluation hors-ligne. Les temps de calcul indicatifs du système ont été réalisés sur un ordinateur portable de 2005 avec le système d'exploitation Windows XP et un processeur cadencé à 1.73 GHz.

| Mesure de qualité | Exercice 1<br>26 diagrammes | Exercice 2<br>30 diagrammes | Exercice 3<br>26 diagrammes |
|---|---|---|---|
| Précision = 1 | 25 | 30 | 10 |
| 0.85 ≤ Précision < 1 | 1 | 0 | 14 |
| 0.7 ≤ Précision < 0.85 | 0 | 0 | 2 |
| Rappel = 1 | 25 | 28 | 1 |
| 0.85 ≤ Rappel <1 | 0 | 2 | 15 |
| 0.7 ≤ Rappel < 0.85 | 1 | 0 | 10 |
| F-Mesure = 1 | 25 | 28 | 1 |
| 0.85 ≤ F-Mesure < 1 | 0 | 2 | 21 |
| 0.7 ≤ F-Mesure < 0.85 | 1 | 0 | 4 |
| Overall = 1 | 25 | 28 | 1 |
| 0.85 ≤ Overall < 1 | 0 | 2 | 9 |
| 0.7 ≤ Overall < 0.85 | 0 | 0 | 12 |
| 0.55 ≤ Overall < 0.7 | 1 | 0 | 4 |
| Temps (min-max) | 0,2 – 0,6 s | 1,1 – 2,0 s | 0,5 – 4,0 s |

**Figure 69 : Résultats de l'évaluation hors-ligne d'ACDC**

Les résultats du système montrent tout d'abord que sur tous les diagrammes comparés, plus de 70% des appariements fournis sont conformes à ceux attendus quels que soient les diagrammes comparés. Le système ACDC commet des erreurs et des oublis mais les appariements de plus de 90% des diagrammes sont pertinents à plus de 85%

(cf. résultats de la *F-Mesure*). Pour 80% des diagrammes traités, les efforts nécessaires pour corriger l'appariement sont mimines (valeur d'*Overall* supérieure à 0,85). La qualité du diagnostic est relativement bonne sur des problèmes simples et moyens (diagrammes des deux premiers exercices). Elle peut cependant être corrigée et améliorée pour des problèmes plus complexes (diagrammes du troisième exercice). Notamment, plus de 85% des résultats en moyenne sont conformes à ceux attendus pour plus de 75% des diagrammes comparés pour le dernier exercice. Les efforts nécessaires pour corriger les erreurs et les oublis de l'ACDC sont plus importants dans le dernier exercice : pour un peu moins de 40% des diagrammes appariés, l'*Overall* est supérieur à 0,85. Néanmoins, pour 85% des diagrammes, la valeur d'*Overall* est supérieure à 0,7 (un résultat encore très acceptable dans le domaine de l'appariement de schémas).

Les résultats plus faibles du système correspondent à des diagrammes qui différent beaucoup des diagrammes de référence. Dans ce cas, le système commet notamment des erreurs d'étiquetage sur certains appariements (mauvais type de différences), relève des appariements parasites dus le plus souvent à la redondance de certains espaces de nommage employés par l'apprenant. De plus, il peut identifier des appariements « structurellement corrects » mais qui ne sont pas les plus intéressants pour le diagnostic. Le système actuel n'est pas capable d'apparier des classes portant des noms différents mais représentant le même concept si la structure des diagrammes est altérée.

Le temps de calcul d'ACDC dépend de la taille du problème et de la distance entre les diagrammes à comparer : les temps de calcul actuels sont acceptables pour pouvoir fournir des rétroactions pédagogiques au cours de l'activité de modélisation.

# Conclusions et perspectives

Dans cette thèse, nous avons présenté un travail ayant pour objectif l'analyse des productions de l'apprenant dans le cadre de l'apprentissage de la modélisation et plus particulièrement dans le contexte de l'activité de construction de diagrammes de classes UML de niveau analyse par des étudiants novices. Les apports de nos travaux se situent à la fois au niveau de la proposition de la méthode d'appariement pour l'analyse des modèles et au niveau de l'application de cette méthode pour le diagnostic et la production de rétroactions synchrones dans l'environnement Diagram.

## Apports des travaux

La méthode d'appariement proposée est inspirée des méthodes actuelles d'appariement de modèles (*ontology matching* et *schema matching*) vus comme des graphes. Nous avons montré qu'il est possible de définir un système d'appariement hybride de niveau-élément et de niveau-structure permettant d'analyser le diagramme de classes UML construit par un apprenant novice dans un contexte d'apprentissage des concepts de la modélisation. Nous avons notamment proposé de prendre en compte explicitement la structure des modèles à comparer avec l'introduction des notions de motifs simples et complexes. Les méthodes d'analyse des modèles des autres EIAH (DesignFirst-ITS, Kermit et Collect-UML) ne tiennent pas compte explicitement de la structuration générale des diagrammes modélisés. Dans notre cas, la structuration des diagrammes dirige la comparaison et l'appariement des éléments. Par rapport aux méthodes d'appariement de modèles classiques, nous qualifions de manière plus précise les appariements à l'aide de la taxonomie des différences que nous avons définie. Nous qualifions les correspondances univoques et multivoques avec des différences. La taxonomie des différences structurelles est suffisamment générale pour s'appliquer à d'autres types de modèles (structurés sous forme de hiérarchies notamment). Nous avons implanté notre système ACDC avec les standards actuels de la communauté UML : le plugin UML2 pour manipuler les diagrammes de classes UML et le format XMI pour les représenter et sauvegarder les diagrammes (avec l'extension .uml2). Ainsi, notre système respecte la norme actuelle UML 2.x et nous pouvons utiliser le système ACDC sur des diagrammes provenant d'autres éditeurs UML standards.

L'application pour le diagnostic et la production de rétroactions synchrones dans Diagram permet de dépasser certaines limites des propositions CBM et *Curriculum*. Notamment, notre méthode ne se concentre pas directement sur les erreurs de l'apprenant qui peuvent se révéler difficiles à identifier étant donné que la tâche de modélisation est ouverte et qu'il peut exister plusieurs solutions pour un même énoncé. Dans les systèmes reposant sur CBM, un modèle construit par l'apprenant est considéré comme correct si aucune contrainte n'a été violée. Dans DesignFirst-ITS, le système exprimera son incapacité à analyser une action de l'apprenant s'il ne peut apparier l'élément concerné par cette action. Notre méthode a l'avantage d'exprimer dans tous les cas de figure des résultats en termes de différences générales qui ne sont pas spécifiques au domaine d'UML. Notre proposition de diagnostic est indépendante des choix pédagogiques de Diagram (les rétroactions pédagogiques). Le diagnostic peut ainsi s'adapter et être utilisé pour différents types de besoins pédagogiques. Enfin notre méthode est adaptée à un EIAH ouvert : nous avons moins de contraintes au niveau des énoncés et des diagrammes de référence que dans les environnements DesignFirst-ITS, Kermit et Collect-UML. L'organisation et le contenu des phrases de l'énoncé ne sont pas contraints explicitement dans notre proposition. Nous ne contraignons pas non plus l'interaction avec l'apprenant lors de la création des noms des différents éléments du modèle. Notre méthode peut ainsi gérer le fait que des éléments soient implicites dans l'énoncé et que certains concepts ne soient pas représentés dans le diagramme de référence.

## Limites et perspectives

Nous commençons par évoquer les limites des propositions sur l'ACDC pour en dégager des perspectives, puis nous terminons par des perspectives plus générales.

Une première limite de notre méthode se situe au niveau des algorithmes retenus et implantés pour la comparaison et l'appariement des modèles dans ACDC : le recours à un algorithme complet de comparaison des motifs des deux diagrammes implique que le temps de calcul va croître en fonction du nombre d'éléments des diagrammes à comparer. La phase d'évaluation des similarités et des différences de chaque couple de motifs pourrait donc être « trop longue » pour des besoins pédagogiques synchrones sur des diagrammes de grande taille (plus d'une centaine d'éléments). Dans notre cas, cette limite n'est pas perceptible car nous ne sommes amenés à ne comparer que des diagrammes de moins de cent éléments (les étudiants sont des novices). La mise en place d'une phase de pré-comparaison partielle orientée spécifiquement sur les espaces de nommage et sur le voisinage des motifs contenus dans les diagrammes peut réduire significativement la complexité de la phase d'évaluation des similarités et des différences : les comparaisons « superflues » lorsque les motifs comparés sont trop distants ne seraient pas réalisées car inutiles pour l'appariement final. La deuxième limite au niveau du mode de fonctionnement retenu est relative à l'appariement glouton n'autorisant pas de *backtracking* lors de la sélection et de la détermination des appariements. Même en se focalisant sur les motifs complexes et simples, le processus reste non déterministe et le résultat produit peut ne pas être le meilleur et nécessiter d'être affiné. Pour dépasser cette limite, il est envisageable de mettre en place un dispositif de correction des appariements déjà identifiés (correction des appariements retenus précédemment en cours d'appariement) ou d'amélioration d'un *mapping* de modèles préalablement constitué comme c'est le cas avec l'algorithme de recherche locale Taboue réactive (cf. partie 4.3.5). La mise en place d'un algorithme de recherche heuristique [Pearl 1984] s'appuyant sur l'importance des motifs et sur les scores de similarité pourrait également permettre de dépasser cette limite.

Il ressort de notre première évaluation hors-ligne (cf. partie 7.2) que les résultats produits par ACDC sont corrects. Néanmoins, le système commet des erreurs, ne repère pas certaines correspondances et peut proposer des appariements trop complexes (inutiles ou inadaptés lors des rétroactions). Dans le cadre de l'environnement Diagram (avec le module de diagnostic et la production des rétroactions pédagogiques), une expérimentation en situation réelle d'apprentissage a été menée à l'automne 2008 lors de quatre séances de TP de trois heures sur un public de dix-huit étudiants novices de DEUST ISR ayant pour consigne de réaliser seize exercices. Les sessions ont été enregistrées et analysées ensuite du point de vue des rétroactions pédagogiques pour évaluer leurs effets sur l'apprenant [Alonso & Py 2009]. Il ressort de cette analyse partielle (i.e. seuls les messages lus par l'apprenant ont été gardés) que 11,6% des messages de rétroaction lus sont incohérents. Les observations de cette expérimentation tendent à renforcer nos premiers constats sur les résulats et les manquements du système ACDC lors de l'évaluation hors-ligne (une analyse plus approfondie des résultats de la dernière expérimentation dans Diagram nous permettra d'améliorer ACDC). Le premier point à améliorer tient à ce que le système ACDC est limité actuellement au niveau de l'appariement de diagrammes distants qui constituent des alternatives correctes ou lorsque plusieurs erreurs interdépendantes altèrent fortement la structure du diagramme construit par l'apprenant. Le second point à améliorer est l'appariement des noms des éléments car il est biaisé en cas d'emploi de noms trop similaires (par exemple : avion, porte-avion, avion de

chasse...) ou différents (pas de similarité directe mais une potentielle synonymie) par rapport à ceux de la solution. De plus, les symétries en termes de structure dans les hiérarchies modélisées peuvent engendrer des appariements non optimums, notamment lorsque l'apprenant supprime un élément d'une hiérarchie ou le déplace : ACDC peut être amené par exemple à repérer le transfert d'une relation vers une classe frère (un appariement correct structurellement) au lieu de repérer un transfert vers un classe fille dans une hiérarchie d'héritages (un appariement plus opportun en réalité). Pour résoudre ces différents problèmes, nous envisageons plusieurs solutions à explorer :

- **Pour évaluer la pertinence des résultats produits** : la qualification de la pertinence des appariements produits peut être gérée en mettant en place un « indice de qualité » permettant de préciser le degré de certitude de chaque appariement produit. Ainsi, ACDC fournirait en sortie uniquement ce dont il est sûr (les appariements qualifiés par un indice supérieur à un certain seuil). Une autre possibilité déjà utilisée dans Design-First ITS (cf. partie 3.3.1) est de focaliser l'appariement en premier lieu sur des éléments requis dans le diagramme de l'apprenant et ensuite sur des éléments optionnels si nécessaires (les éléments optionnels pouvant être représentés par des alternatives correctes par l'apprenant).
- **Pour pallier le problème particulier d'appariement des espaces de nommage** : la voie nous paraissant la plus pertinente est le recours à une « base de synonymes » comme c'est également le cas dans Design-First. Nous n'envisageons pas la contrainte et la simplification des énoncés employées dans Kermit, Collect-UML et Design-First et la « focalisation » sur les noms de l'énoncé lors de l'interaction dans Kermit, Collect-UML car ce sont des limites que nous souhaitons dépasser.

Le paramétrage de la mesure de similarité et des appariers du système ACDC peuvent se révéler ardu. Il ressort qu'un unique paramétrage ne peut s'appliquer à tous les modèles car certains choix sont exclusifs et orientent les résultats. Par exemple, les critères propres aux noms des éléments, lorsqu'ils ont un poids important, limitent l'appariement des éléments implicites et la prise en compte de la structure. Inversement, si ces critères sont dépréciés, l'appariement se concentre sur la structure et néglige certains appariements pouvant être pertinents car leurs noms sont relativement proches. Une autre difficulté pour le système est d'être à la fois capable de repérer des transferts de relation d'un même type et des changements de type de relation sans transfert : ces deux problèmes sont exclusifs et impossibles à prendre en compte de manière complète quel que soit le paramétrage d'ACDC. Actuellement, le système est paramétré de telle manière qu'il peut identifier ces deux différences mais produit souvent des appariements « parasites ». Une première possibilité pour dépasser ces limites serait de définir un système capable d'étudier automatiquement la structure du diagramme de référence (et l'énoncé associé) pour ensuite paramétrer certains critères critiques de l'ACDC (cette solution est compliquée à concevoir et à mettre en place). Plus simplement, la conception d'une interface de paramétrage explicite et/ou d'un questionnaire[30] posé au concepteur de l'exercice pourrait permettre de paramétrer le système ACDC et d'améliorer ses résultats.

Une dernière limite est la prise en compte d'un seul diagramme de référence par ACDC. Le système n'est donc pas capable de prendre en compte des solutions alternatives correctes trop éloignées de celle de référence. Une première solution pour répondre partiellement à ce problème est de repérer des *patterns* d'analyse et/ou de conception dans les

[30] Une question posée pour gérer le problème des classes implicites pourrait être de la forme : « Le diagramme de référence représente-t-il une ou plusieurs classes implicites à l'énoncé ? (Si, oui lesquelles ?) ». Cette interface de paramétrage explicite pourrait améliorer l'appariement des noms si le concepteur peut associer une liste de noms alternatifs aux éléments du diagramme (actuellement, seule la liste des expressions de l'énoncé liées aux éléments du modèle de référence est utilisée pour identifier des noms alternatifs aux éléments).

diagrammes à comparer car ils permettent de déduire des constructions équivalentes sémantiquement mais différentes structurellement. Une autre solution pour répondre à ce problème est de comparer le diagramme de l'apprenant à plusieurs diagrammes de référence, qu'ils soient corrects ou erronés et complets ou partiels. La prise en compte de plusieurs références apporte une difficulté supplémentaire :

*Comment décrire et qualifier les diagrammes les uns par rapport aux autres ?*

Si les diagrammes sont exclusifs structurellement mais « sémantiquement » équivalents (c'est rarement le cas pour des diagrammes complets car des *patterns* sont appliqués en général sur une partie d'un diagramme), il n'y a pas difficulté majeure car il suffit de choisir le diagramme de l'apprenant qui s'apparie le mieux à l'une ou l'autre des alternatives. Dans les autres cas de figure tels que des diagrammes pouvant avoir des parties communes ou être une partie alternative d'un autre diagramme (i.e. un problème d'appariement de sous-graphes), il est très difficile de qualifier explicitement des relations explicites et précises entre les références. L'ACDC serait capable actuellement de comparer un diagramme à plusieurs autres diagrammes et de choisir à l'issue de la phase d'évaluation des similarités et des appariements locaux lesquels des diagrammes s'apparient le « mieux » en prenant en compte le score de similarité général propre aux diagrammes comparés un à un. Nous n'avons pas expérimenté cette propriété du système, étant donné qu'elle ne résout pas de manière complète le problème de la prise en compte de solutions multiples lors de l'analyse des productions de l'apprenant.

## Perspectives générales

La méthode d'appariement définie et le système ACDC peuvent être réutilisés (et adaptés) sur d'autres types de modèles. Un point intéressant à explorer est l'application de notre méthode à d'autres types de modèles structurés où le formalisme est défini. Notamment, la méthode pourrait être réutilisée de manière « directe » sur d'autres modèles statiques tels que les modèles Entité-Relation. Ces derniers ont des caractéristiques très proches des diagrammes de classes UML et peuvent être considérés comme un sous-ensemble des diagrammes de classes UML. Des travaux ont notamment été réalisés pour transformer de manière automatique un diagramme Entité-Relation sous forme d'un diagramme de classes UML (en définissant des règles de transformation issues de l'étude de leurs deux métamodèles). Dans ce cas, il serait donc envisageable dans un premier temps de convertir les modèles Entité-Relation en diagrammes de classes UML pour ensuite les apparier. Pour d'autres types de modèles plus éloignés des spécificités des diagrammes de classes UML (d'autres types de diagrammes UML, des modèles de tâches ou des modèles de circuits électroniques [Tanana *et al.* 2008] par exemple), il faudrait réaliser une étude plus approfondie des modèles, de leurs métamodèles associés pour qualifier les types des motifs (simples et complexes), leurs spécificités propres et vérifier l'adéquation des choix de notre méthode et notamment l'application des différences proposées dans notre méthode.

Nous avons appliqué notre méthode pour le diagnostic et la production de rétroactions pédagogiques pour l'apprenant dans Diagram. Étant donné que notre proposition est définie de manière indépendante des choix pédagogiques de cette application (nos résultats sont convertis en différences pédagogiques après avoir été produits), nous pouvons envisager d'autres types d'application dans le contexte de l'apprentissage en nous focalisant sur l'enseignant et ses besoins spécifiques. Une difficulté récurrente pour l'enseignant est l'activité de correction et d'évaluation des modèles élaborés par les apprenants. Cette activité est longue et fastidieuse car elle demande à

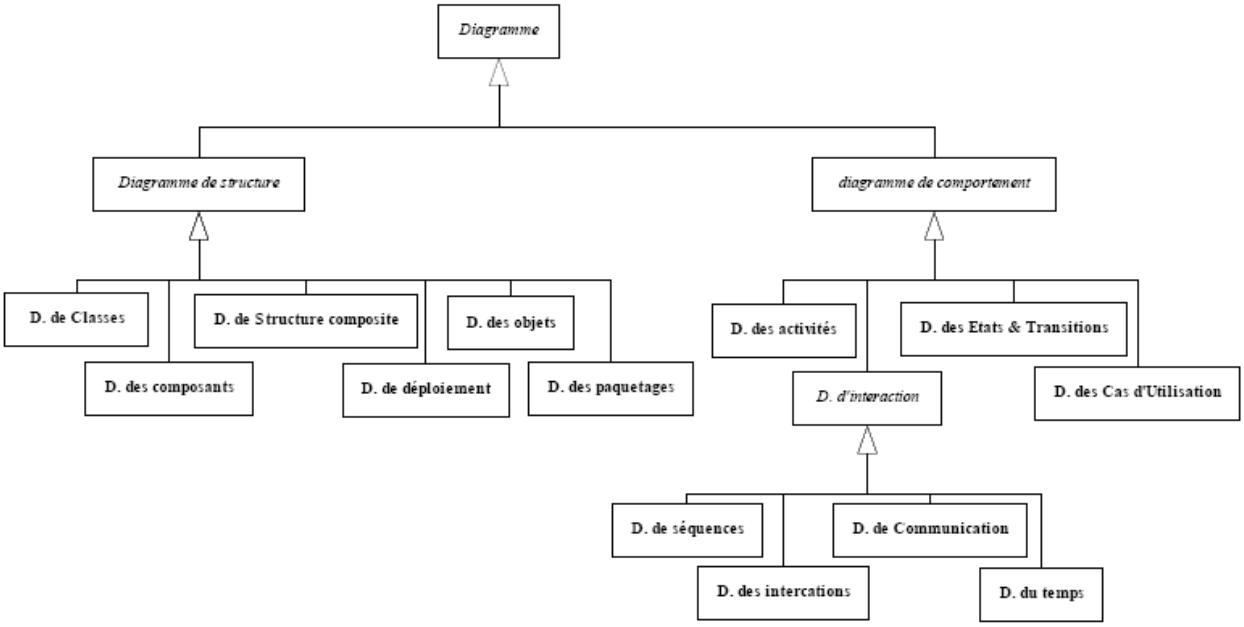
l'enseignant d'analyser un par un les diagrammes pour essayer de comprendre le sens modélisé et de repérer les « mauvais choix de modélisation ». Il est important de remarquer que l'enseignant n'utilise pas nécessairement une solution de référence et que chaque enseignant peut analyser différemment les modèles en exprimant par exemple des « règles » sur les éléments à ne pas représenter, les éléments requis ou des *patterns* à utiliser... Les résultats produits par notre système (les correspondances et les différences identifiées) peuvent néanmoins être intéressants car il apporte de manière automatique une vue complète et précise sur des différences par rapport à un diagramme « attendu ». Cette application demande cependant une étude approfondie des attentes et des besoins de l'enseignant. L'adaptation d'ACDC pour aider l'enseignant dans cette tâche nécessiterait de filtrer, de regrouper et de présenter les différences en fonction de ses besoins dans une interface spécifique dédiée à l'enseignant.

Les résultats de nos travaux peuvent également constituer une première base pour le suivi des apprenants par l'enseignant au cours ou à l'issue de l'activité de modélisation. En analysant, avec ACDC, les différents diagrammes construits par un même apprenant pour différents exercices, un outil de suivi pourrait se focaliser sur certains types de différences, les filtrer, les qualifier et ensuite les présenter à l'enseignant. La présentation d'une vue synthétique des différences qu'un groupe d'apprenants produit est également envisageable car ACDC peut traiter en parallèle automatiquement plusieurs diagrammes de classes UML pour un même exercice : un outil peut donc être lié en sortie pour synthétiser et convertir les résultats d'ACDC. Une possibilité d'utilisation serait de relever et quantifier les différences les plus identifiées pour tel exercice (par exemple l'élément « Entreprise » a été introduit par sept apprenants, la classe implicite « Crayon » n'a pas été représentée par six apprenants).

# Annexes

# LISTE DES ANNEXES

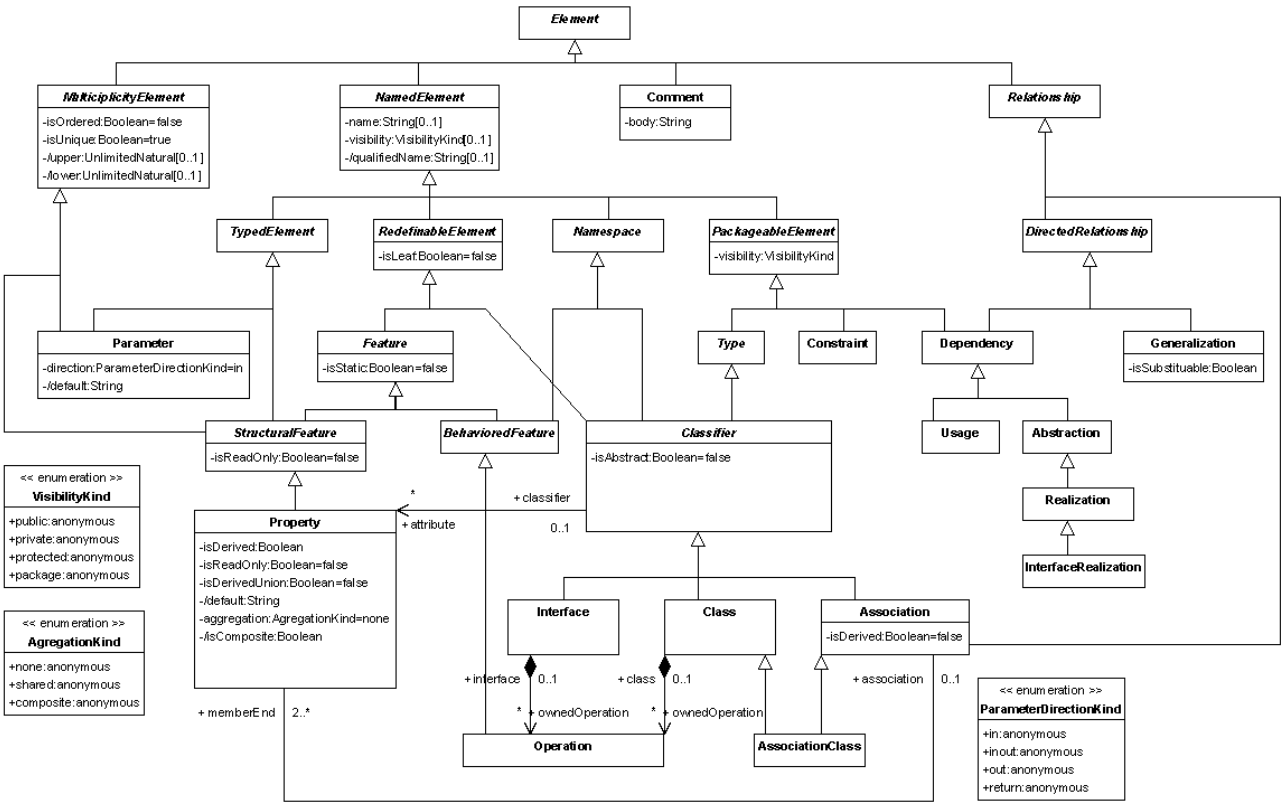
# Annexe 1 : Hiérarchie des diagrammes proposés dans UML 2.x



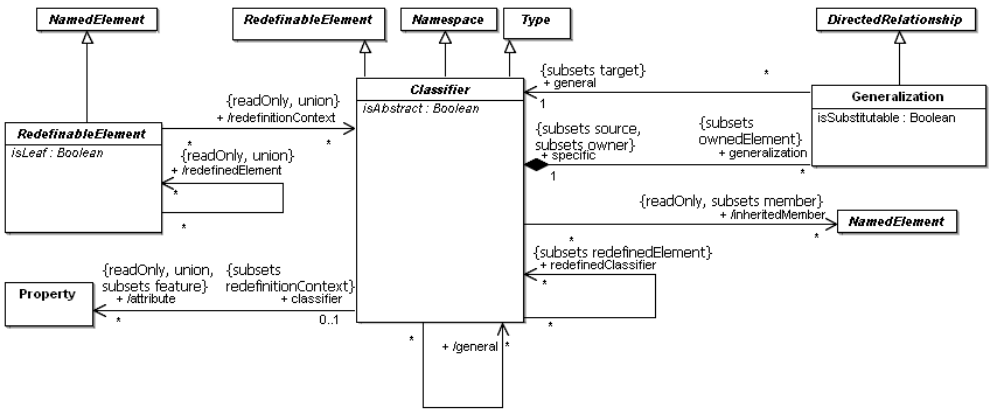

Cette hiérarchie des diagrammes proposés dans UML est extraite de [Modelique].

# Annexe 2 : Extrait du métamodèle UML 2.x des diagrammes de classes

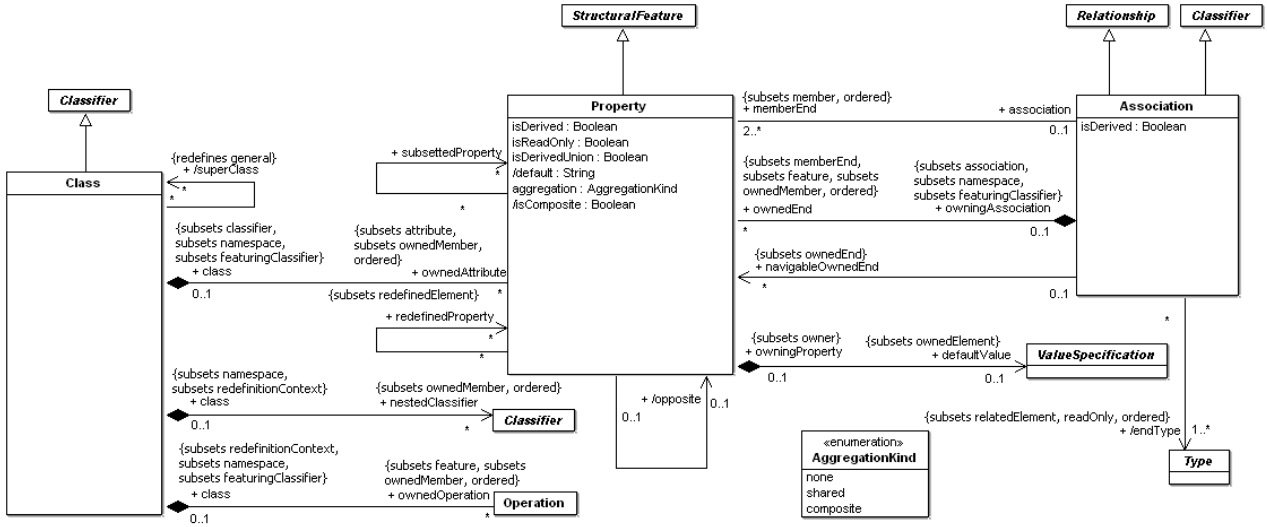
Dans cette annexe, nous exposons quelques diagrammes de classes UML sur lesquels a reposé notre analyse du métamodèle des diagrammes de classes UML 2.x. ils sont issus de la documentation définie par l'OMG [OMG UML] à chaque nouvelle version ou sous-version d'UML. Nous nous sommes plus penchés sur le document décrivant la superstructure d'UML (les constructions de niveau utilisateur du métamodèle) [UML Superstructure]. Le diagramme de classes UML ci-dessous est un extrait du métamodèle UML 2.x représentant une vue générale des éléments fondamentaux intervenant dans un diagramme de classes tels qu'il ont été définis par l'OMG. Nous avons utilisé l'éditeur UML Poséidon pour le représenter.



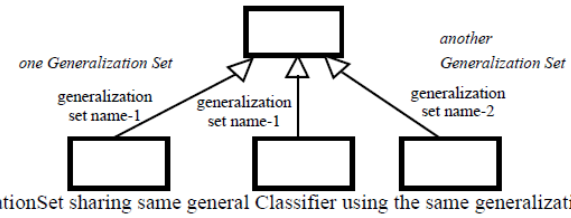

La superstructure précise les caractéristiques et liens entre les différents éléments intervenant dans les diagrammes UML tels que le classificateur dans le diagramme de classes suivant :



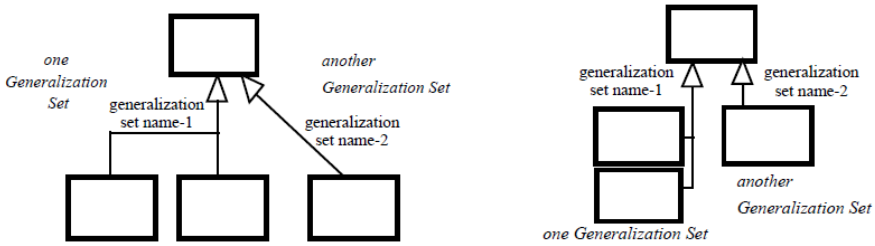
Ce second diagramme extrait lui-aussi directement de la superstructure est centré sur les liens et les caractéristiques des propriétés UML et notamment sur leur rôle de ponts entre les classes et les associations dans la version 2.x du métamodèle d'UML.



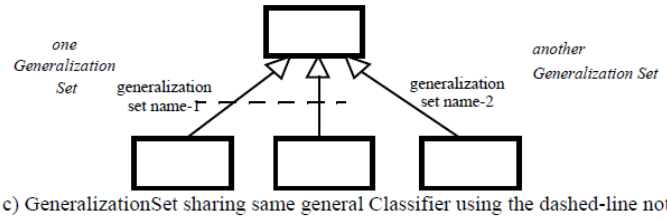

Enfin ce dernier ensemble de diagrammes explique certaines spécificités de représentation des relations d'héritage (*generalization*) sous de forme de *GeneralizationSet* en recourant à des exemples de hiérarchies de classes :




a) GeneralizationSet sharing same general Classifier using the same generalization relationship names.




b) GeneralizationSet designation by subtypes sharing a common generalization arrowhead.




c) GeneralizationSet sharing same general Classifier using the dashed-line notation.

# Annexe 3 : Règles de cohérence des diagrammes de classes

Cet extrait des règles de cohérence de [Seuma Vidal *et al.* 2005] définit la syntaxe et la cohérence de quelques éléments représentables dans le diagramme de classes UML 2.0 qui nous intéresse plus particulièrement dans notre travail :

**Règles pour les espaces de nommage :**

Règle 4 : Tous les membres d'un espace de nommage doivent pouvoir être distingués.
Remarque : Par défaut, un élément A se distingue d'un élément B si A n'est pas du même type que B, un sous-type de B, ou, si A et B n'ont pas le même nom. Cette définition du qualificatif <distingué> peut être redéfinie, c'est notamment le cas pour les opérations qui sont distinguables en fonction de leur signature.

**Règles pour les multiplicités :**

Règle 13 : L'expression d'une multiplicité doit suivre la syntaxe :
multiplicity ::= <multiplicity_range> ['{'<order_designator> [',' <uniqueness_designator>]'}']
<multiplicity_range> ::= [ <lower> '..' ] <upper>
<lower> ::= integer | <value_specification>
<upper> ::= * | <value_specification>
<order_designator> ::= 'ordered' | 'unordered'
<uniqueness_designator> ::= 'unique' | 'nonunique'

**Règles pour les contraintes :**

Règle 23 : L'expression d'une contrainte doit suivre la syntaxe suivante :
constraint ::= '{'[<name>':'] <boolean expression>'}'

**Règles pour les propriétés :**

Règle 32 : Excepté les propriétés subsets et redefines, les propriétés doivent respecter la syntaxe :
property ::= {name ['=' value]}
où :
- name est le nom de la propriété (le marqueur de la propriété) ;
- le symbole '=' est un séparateur ;
- value est la valeur (celle du marqueur) à laquelle doit se trouver la propriété ; ce champ est optionnel.

**Règles pour les classes :**

Règle 44 : Une classe est forcément nommée.

**Règles pour les opérations :**

Règle 55 : L'expression d'une opération doit suivre la syntaxe suivante :
[visibility] name ( parameter-list ) : property-string

Règle 56 : Le champ parameter-list de la règle 55 doit suivre la syntaxe :
direction name : type-expression ['multiplicity']' = default-value [{ property-string }]
où direction a comme valeur par défaut in

**Règles pour les attributs :**

Règle 76 : Les attributs doivent être présentés en suivant la syntaxe suivante :
[visibility][/]name[:type][multiplicity][= default] [{ property-string }]

Règle 77 : Un attribut doit avoir un nom.

**Règles pour les classes d'association :**

Règle 96 : Une classe d'association ne peut pas être définie entre elle-même et quelque chose d'autre. Il ne faut pas non plus que la classe d'association soit un parent ou un enfant d'une des classes mises en jeu.

Règle lien 8 : Les classes association étant des associations, elles doivent respecter les propriétés des associations.

Règle lien 9 : Les classes association étant des classes, elles doivent respecter les propriétés des classes présentées.

**Règles pour les relations d'héritage :**

Règle 130 : Les éléments mis en relation par une relation d'héritage doivent être des classificateurs.

Règle 131 : La hiérarchie des généralisations doit être orientée (*directed* en anglais) et acyclique.
Remarque : Ainsi, un classificateur ne doit pas être à la fois un ancêtre et un descendant d'un même classificateur.

Règle 132 : Une relation d'héritage doit mettre en jeu exactement un parent et un enfant.

**Règles pour les éléments abstraits :**

Règle 142 : Un élément abstrait doit être l'ancêtre (direct ou non) d'au moins un élément non abstrait.

Règle 143 : Un élément abstrait ne doit pas avoir d'instance.

**Règles pour toutes les associations :**

Règle 173 : Si l'association est une classe d'association, le nom optionnel de l'association doit être le même que celui de la classe d'association.

Règle 174 : Une association met en relation des classificateurs exclusivement.

Règle 175 : Toute association a au moins deux possessions (qui au niveau modèle sont représentées par des fins d'association).

Règle 176 : Toute possession qui est une fin d'association doit avoir un type et ce type doit être un classificateur. D'un point de vue modèle ceci veut dire qu'une fin d'association est forcément reliée à un classificateur.
Remarque : Il est possible qu'une association mette en jeu uniquement un classificateur avec lui-même.

Règle 178 : Si une fin d'association est une classe alors les autres fins de cette association doivent aussi être des classes.

Règle 198 : Dans le cas d'association n-aire o n>2, les agrégations composites ne peuvent pas apparaître.

**Règles pour les agrégations :**

Règle 202 : Pour une association, seule une fin d'association peut avoir une agrégation de type composite ou share.

Règle 204 : Aucun circuit faisant intervenir des liens qui correspondent à des associations dont le type d'agrégation est share ou composite ne doit apparaître au niveau objet.

Règle 205 : Aucun circuit faisant intervenir des associations dont le type d'agrégation est share ou composite et dont la multiplicité est 1 ne doit apparaître.

**Règles pour les compositions :**

Règle lien 14 : La composition est une forme d'agrégation forte et doit donc répondre aux règles exposées pour les agrégations.

Règle 206 : Une multiplicité d'une agrégation composite du coté du classificateur composé ne doit pas avoir sa borne supérieure plus grande que 1.

Règle 207 : Toute instance de la classe composante doit appartenir à une et une seule instance de la classe composée.

Règle 208 : Lorsque plusieurs classes composent une même classe et que ces classes sont mises en relation via une association, les objets (instances des classes composantes) qui sont mis en relation par un lien (instance de l'association) doivent tous appartenir au même objet composite (qui est l'instance de la classe composéee). En d'autres termes, dans ce cas, le lien doit mettre en relation des objets (instances des classes composantes) qui appartiennent au même objet (instance de la classe composée).

Règle 209 : Lorsque deux classes sont reliées par une relation de composition, seul l'objet composé ou un des objets qu'il englobe (par héritage) peuvent créer ou détruire les objets composants.

Règle 210 : La valeur de isComposite est vraie uniquement si aggregation est composite

**Règles pour les diagrammes de classes :**

Règle 220 : Les noeuds graphiques contenus dans un diagramme de classes ne peuvent être que :
- une classe ;
- une interface ;

Règle 221 : Les chemins graphiques pouvant être contenus dans un diagramme de classes ne peuvent être que :
- une association de type agrégation, composite ou simple ;
- une dépendance ;
- une généralisation ;
- une réalisation.

Règle 222 : Le seul nœud graphique pouvant être contenu dans un diagramme de paquetages est le paquetage.

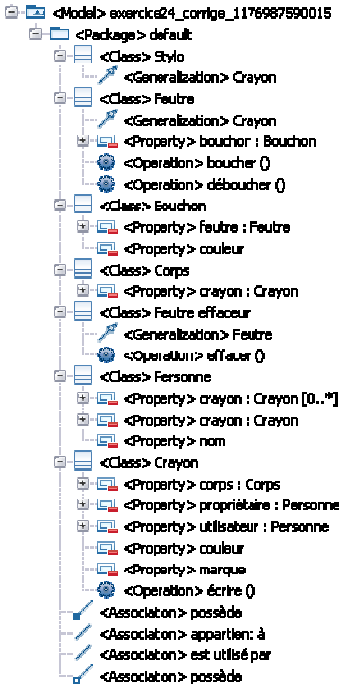
Règle 223 : Les chemins graphiques pouvant être contenus dans un diagramme de paquetages ne peuvent être que :
- une importation de paquetage (*packageImport* en anglais), dépendance avec le mot clé <import> ou <access> ;
- une importation d'un élément (*elementImport* en anglais), dépendance avec le mot clé <import> ;
- un interclassement de paquetage (*packageMerge* en anglais), dépendance avec le mot clé <merge> ;
- une dépendance ;

Règle 224 : Le seul nœud graphique pouvant être contenu dans un diagramme d'objets est une instance de spécification.

Règle 225 : Les seuls chemins graphiques pouvant être contenus dans un diagramme d'objets ne peuvent être que des liens.

# Annexe 4 : Exemple de fichier .uml2 (XMI) pour le modèle idéal de l'exercice « Stylo et Feutre »

Les diagrammes manipulés par l'ACDC sont chargés et sauvegardés au format XMI dans des fichiers portant l'extension *.uml2*. Ils sont lisibles graphiquement et modifiables directement dans l'éditeur EMF [EMF] d'Eclipse. L'arborescence ci-dessous correspond au diagramme de référence de l'exercice « Stylo et Feutre » (cf. figure 11) :

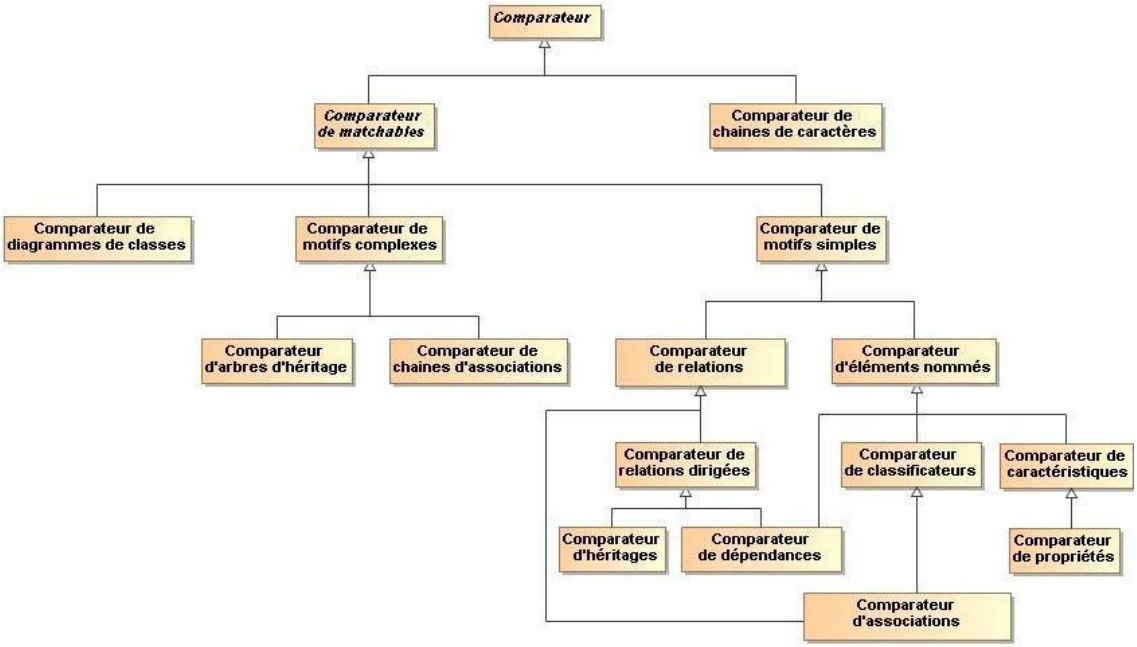



Le contenu brut du fichier *.uml2* sauvegardant le diagramme de référence de l'exercice « Stylo et Feutre » est le suivant :


```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1" xmlns:uml="http://www.eclipse.org/uml2/2.1.0/UML" xmi:id="_3T33W051EduS3qE78T5xzg" name="
  <packagedElement xmi:type="uml:Package" xmi:id="_3UKrw051EduS3qE78T5xzg" name="default">
    <packagedElement xmi:type="uml:Class" xmi:id="_3WPvY051EduS3qE78T5xzg" name="Stylo">
      <generalization xmi:id="_3bnXn-51EduS3qE78T5xzg" general="_3WZgV-51EduS3qE78T5xzg"/>
    </packagedElement>
    <packagedElement xmi:type="uml:Class" xmi:id="_3WPvY051EduS3qE78T5xzg" name="Feutre">
      <generalization xmi:id="_3bnXmu51EduS3qE78T5xzg" general="_3WZgV-51EduS3qE78T5xzg"/>
      <ownedAttribute xmi:id="_3bnXoe51EduS3qE78T5xzg" name="" visibility="private" type="_3WPvvu51EduS3qE78T5xzg" aggregation="shared" association="_3bnXo051EduS3qE
        <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_QfKs4KG4Edy5kZejLXHHyQ" value="1"/>
        <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_RBZwrKG4Edy5kZejLXHHyQ"/>
      </ownedAttribute>
      <ownedOperation xmi:id="_3cNNc051EduS3qE78T5xzg" name="toucher"/>
      <ownedOperation xmi:id="_3cNNce51EduS3qE78T5xzg" name="déboucher"/>
    </packagedElement>
    <packagedElement xmi:type="uml:Class" xmi:id="_3WPvVu51EduS3qE78T5xzg" name="Bouchon">
      <ownedAttribute xmi:id="_3bnXp051EduS3qE78T5xzg" name="" visibility="private" type="_3WPvVe51EduS3qE78T5xzg" association="_3bnXo051EduS3qE78T5xzg">
        <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_58YdEKG4Edy5kZejLXHHyQ" value="1"/>
        <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_KFiYYKG4Edy5kZejLXHHyQ"/>
      </ownedAttribute>
      <ownedAttribute xmi:id="_3bnXp-51EduS3qE78T5xzg" name="couleur" visibility="private"/>
    </packagedElement>
    <packagedElement xmi:type="uml:Class" xmi:id="_3WZgY051EduS3qE78T5xzg" name="Corps">
      <ownedAttribute xmi:id="_3bA6605 1EduS3qE78T5xzg" name="" visibility="private" type="_3WZgV-51EduS3qE78T5xzg" association="_3WsbU051EduS3qE78T5xzg">
        <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_3bA6ou51EduS3qE78T5xzg" value="1"/>
        <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_3bA8oe51EduS3qE78T5xzg" value="1"/>
      </ownedAttribute>
    </packagedElement>
    <packagedElement xmi:type="uml:Class" xmi:id="_3WZgYe51EduS3qE78T5xzg" name="Feutre_effaceur">
      <generalization xmi:id="_3WZgZ051EduS3qE78T5xzg" general="_3WPvVe51EduS3qE78T5xzg"/>
      <ownedOperation xmi:id="_3cNNcu51EduS3qE78T5xzg" name="effacer"/>
    </packagedElement>
    <packagedElement xmi:type="uml:Class" xmi:id="_3WZgYu51EduS3qE78T5xzg" name="Personne">
      <ownedAttribute xmi:id="_3bnXlO51EduS3qE78T5xzg" name="" visibility="private" type="_3WZgY-51EduS3qE78T5xzg" association="_3bnXk051EduS3qE78T5xzg">
        <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_3bnXlu51EduS3qE78T5xzg" value="*"/>
        <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_3bnXle51EduS3qE78T5xzg"/>
      </ownedAttribute>
      <ownedAttribute xmi:id="_3bnXm-51EduS3qE78T5xzg" name="" visibility="private" type="_3WZgV-51EduS3qE78T5xzg" association="_3bnXl-51EduS3qE78T5xzg">
        <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_3bnXme51EduS3qE78T5xzg" value="1"/>
        <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_3bnXm051EduS3qE78T5xzg"/>
      </ownedAttribute>
      <ownedAttribute xmi:id="_3bnXq051EduS3qE78T5xzg" name="rom" visibility="private"/>
    </packagedElement>
    <packagedElement xmi:type="uml:Class" xmi:id="_3WZgY-51EduS3qE78T5xzg" name="Crayon" isAbstract="true">
      <ownedAttribute xmi:id="_3aI3O051EduS3qE78T5xzg" name="" visibility="private" type="_3WZgV051EduS3qE78T5xzg" aggregation="composite" association="_3WsbU051EduS
        <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_3aI3vs051EduS3qE78T5xzg" value="1"/>
        <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_3aI_s051EduS3qE78T5xzg" value="1"/>
      </ownedAttribute>
      <ownedAttribute xmi:id="_3bnXke51EduS3qE78T5xzg" name="propriétaire" visibility="private" type="_3WZgYu51EduS3qE78T5xzg" association="_3bnXk051EduS3qE78T5xzg">
        <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_3bnXk-51EduS3qE78T5xzg" value="1"/>
        <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_3brXku51EduS3qE78T5xzg" value="1"/>
      </ownedAttribute>
      <ownedAttribute xmi:id="_3bnXm051EduS3qE78T5xzg" name="utilisateur" visibility="private" type="_3WZgYu51EduS3qE78T5xzg" association="_3bnXl-51EduS3qE78T5xzg">
        <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_3bnXmu51EduS3qE78T5xzg" value="1"/>
        <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_3brXme51EduS3qE78T5xzg"/>
      </ownedAttribute>
      <ownedAttribute xmi:id="_3bxIk051EduS3qE78T5xzg" name="couleur" visibility="private"/>
      <ownedAttribute xmi:id="_3bxIke51EduS3qE78T5xzg" name="marque" visibility="private"/>
      <ownedOperation xmi:id="_3cNNc-51EduS3qE78T5xzg" name="écrire"/>
    </packagedElement>
    <packagedElement xmi:type="uml:Association" xmi:id="_3Wsbl051EduS3qE78T5xzg" name="possède" memberEnd="_3aI3O051EduS3qE78T5xzg _3bA6o051EduS3qE78T5xzg"/>
    <packagedElement xmi:type="uml:Association" xmi:id="_3bnXk051EduS3qE78T5xzg" name="appartient à" memberEnd="_3bnXke51EduS3qE78T5xzg _3bnXlO51EduS3qE78T5xzg"/>
    <packagedElement xmi:type="uml:Association" xmi:id="_3bnXl-51EduS3qE78T5xzg" name="est utilisé par" memberEnd="_3bnXm051EduS3qE78T5xzg _3bnXm-51EduS3qE78T5xzg"/>
    <packagedElement xmi:type="uml:Association" xmi:id="_3bnXo051EduS3qE78T5xzg" name="possède" memberEnd="_3bnXoe51EduS3qE78T5xzg _3bnXp051EduS3qE78T5xzg"/>
  </packagedElement>
</uml:Model>
```

# Annexe 5 : Hiérarchie de comparateurs d'ACDC

Les comparateurs prennent en charge l'évaluation des similarités et des différences locales aux modèles, aux motifs, et aux chaines de caractères à apparier. La hiérarchie de comparateurs suivante permet de mener la comparaison des constituants des diagrammes de classes UML :




Tous les comparateurs ont une structure et un fonctionnement général commun pour évaluer les similarités et les différences des objets à comparer. Un comparateur est instancié pour chaque ensemble de motifs à comparer. Chaque comparateur mémorise les objets qu'il compare, les scores de similarité calculés et les différences locales identifiées pour éviter de les recalculer lors des comparaisons d'autres objets en dépendant. Si plusieurs objets ont déjà été comparés et qu'ils participent à la confrontation d'autres objets alors le comparateur existant de ces objets est consulté. Au besoin, les scores et les différences locales calculés sont mis à jour lorsque des problèmes de circularité se présentent et que le calcul est partiel.

Le **comparateur de *matchables*** définit le comportement général commun à tous les comparateurs de la méthode ACDC s'appliquant sur les objets appariables (i.e. les modèles et les motifs) sous forme de trois méthodes distinctes nommées respectivement *compare*, *compareSimpleScore* et *compareComplexScore*. La fonction *compare* définit la manière dont les résultats des fonctions *compareSimpleScore* et *compareComplexScore* sont agrégés et dans quel ordre les calculs sont effectués. Les fonctions *compareSimpleScore* et *compareComplexScore* sont définies ou redéfinies dans chaque comparateur individuel en fonction des dimensions qu'ils prennent en compte (les comparateurs en héritant définissent les méthodes *compareSimpleScore* et *compareComplexScore*). La méthode *compareComplexScore* définit comment le contexte de plusieurs objets sont comparés et appariés localement (les objets contenus, conteneurs et liés).

Le c**omparateur de chaînes de caractères** se charge de comparer plusieurs chaînes de caractères et de qualifier leurs différences. Tous les noms des éléments du modèle sont confrontés aux autres noms des éléments d'un autre modèle en invoquant ce comparateur spécifique. Ce comparateur n'évalue pas de score complexe étant donné que les noms ne

dépendent d'aucun autre élément du modèle. Le fonctionnement de la mesure de similarité des noms est précisé dans la partie 5.3.5 et son algorithme est reporté en annexe 6.

Le **comparateur de diagrammes de classes** se focalise sur la comparaison des diagrammes de classes et définit la manière et l'ordre dans lequel les motifs les constituant sont comparés localement. Les choix de parcours des motifs des diagrammes à apparier sont définis dans la méthode *compareComplexScore* de ce comparateur. Nous avons défini plusieurs modes de fonctionnement pour mener la comparaison :

- **Ascendant *(Bottom-Up)*** : les motifs simples sont comparés un à un puis ensuite seulement les motifs complexes sont comparés un à un. Dans ce cas de figure la comparaison des motifs simples n'est pas dirigée explicitement par la structure des motifs complexes. Même si les motifs simples sont comparés par type, les classes, les relations, les caractéristiques ne sont pas comparées dans un ordre précis. La comparaison des motifs complexes utilisent les résultats issus de la comparaison des motifs simples sans réaliser de comparaisons supplémentaires.
- **Descendant *(Top-Down)*** : l'évaluation des similarités et des différences des motifs complexes dirige celle des motifs simples. Seuls les motifs complexes sont comparés un à un dans ce comparateur mais localement à chaque couple de motifs complexes comparés (*via* un comparateur de motifs complexes par exemple), les motifs les constituants sont comparés un à un. Ce mode de fonctionnement est préféré car il permet d'obtenir de meilleurs résultats en dirigeant la comparaison des diagrammes et en contextualisant l'évaluation des motifs simples. Les motifs simples sont comparés dans le contexte de l'évaluation d'un couple de motifs complexes en plus de celui des diagrammes.

Le **comparateur de motifs complexes** évalue les similarités et les différences selon des critères communs à tous les motifs complexes, c'est-à-dire les motifs simples contenus et liés (i.e. des classificateurs et des relations), les motifs complexes contenus (enfants), conteneurs (parents) et liés directement. Le score simple est une somme pondérée et moyennée des différences de nombres de classificateurs contenus, de relations (contenues, internes et externes[31]), de motifs complexes (contenus, conteneurs et liés). Le score complexe est exprimé en fonction des meilleurs appariements locaux univoques des motifs complexes contenus et de ceux des motifs simples contenus et liés. Pour cela, le motif source est tout d'abord comparé aux motifs enfants du motif cible et inversement (pour vérifier si les motifs complexes ne sont pas inclus). Ensuite, les classificateurs contenus sont parcourus et appariés en premier localement au couple de motif complexe comparé puis ce sont les relations contenues, internes et externes qui sont appariées. La valeur de similarité de deux motifs complexes est égale à la somme des scores des motifs complexes et simples appariés localement.

Deux comparateurs héritent du comparateur de motifs complexes pour ceux spécifiques aux diagrammes de classes UML. Le **comparateur d'arbres d'héritages** évalue les similarités et différences de plusieurs arbres d'héritages et prend en compte plus particulièrement leurs nombres de racines, de feuilles, de classes héritant de manière multiple (i.e. héritant de plusieurs classes directement ou en losange) dans le score simple. Le parcours des motifs simples contenus peut être réalisé plus précisément des racines vers les feuilles. Il appelle les méthodes de calcul du comparateur de motifs complexes pour compléter son évaluation. Le **comparateur de chaînes d'associations** se focalise sur le type précis de motif complexe correspondant aux chaînes d'associations. Le score simple cumule les différences de nombres

[31] Une relation interne à un motif complexe est une relation n'appartenant pas au motif complexe mais reliant plusieurs classificateurs y apparaissant. Par exemple, dans un arbre d'héritages, une relation d'association liant deux classes d'une hiérarchie d'héritages est une relation interne. Une relation externe lie un classificateur du motif complexe à un classificateur d'un autre motif complexe.

d'extrémités des chaînes d'associations, de classes liées à plus deux associations, de classes ayant une association réflexive, de classes reliées par plus d'une association. Le parcours des motifs contenus est réalisé d'une extrémité aux autres si les chaînes en possèdent. Ce comparateur appelle les méthodes de calcul du comparateur de motifs complexes pour compléter son évaluation des similarités et des différences.

Le **comparateur de motifs simples** (i.e. les éléments UML dans les diagrammes de classes UML) se focalise uniquement sur la nature (le type) des éléments à comparer. C'est la seule dimension qui peut permettre de différencier tous les éléments présents dans un modèle UML.

Le **comparateur d'éléments nommés** se charge de la comparaison des visibilités des éléments (communes à tous les éléments nommés) et de diriger la comparaison spécifique des espaces de nommage des éléments pouvant être nommés en invoquant un comparateur de chaînes de caractères pour chaque paire de noms à comparer. Il appelle les méthodes du comparateur d'éléments pour compléter ses calculs.

Le **comparateur de classificateurs**, pour évaluer le score simple, se concentre sur l'abstraction des classes, le nombre d'éléments encapsulés, hérités et les relations liées aux classificateurs comparés. Le score complexe est une combinaison des scores des meilleurs appariements locaux des propriétés contenues, des opérations contenues et des relations liées. Pour cela, il invoque de manière séquentielle respectivement des comparateurs de propriétés, de caractéristiques et enfin de relations (relations, relations dirigées, associations en fonction des types). Les relations sont comparées en dernier pour limiter les problèmes de circularité. Les meilleurs appariements univoques sont retenus et sommés dans le score complexe. Il appelle les méthodes du comparateur d'éléments nommés.

Les relations sont comparées à l'aide de trois comparateurs. Le **comparateur de relations** prend en compte les classificateurs liés aux relations en invoquant un comparateur de classificateurs. Les scores des classificateurs appariés deux à deux sont pris en compte dans le calcul du score complexe des relations. Les méthodes de calcul du comparateur d'éléments complètent cette mesure de similarité. Le **comparateur de relations dirigées** prend en compte le sens des relations et le reste de l'évaluation des similarités est menée dans le comparateur de relations. Enfin le **comparateur d'associations** se focalise sur le type d'agrégation des associations et cumule à la fois les résultats des comparateurs de classificateurs et de relations.

Les caractéristiques UML sont gérées par deux comparateurs. Le **comparateur de caractéristiques** prend en compte les similarités et différences des conteneurs à apparier en invoquant un comparateur de classificateurs. Il complète son évaluation à l'aide des méthodes du comparateur d'éléments nommés. Pour les propriétés, l'évaluation est complétée par le **comparateur de propriétés** qui prend en compte les similarités et les différences des types, des multiplicités. Nous n'avons pas défini de comparateur spécifique aux opérations car elles sont définies dans notre cas par un nom (il n'y a pas de paramètres d'entrée ni de sortie). Le comparateur de caractéristiques est donc suffisant pour comparer deux opérations.

# Annexe 6 : Algorithme de mesure des similarités de deux chaînes de caractères

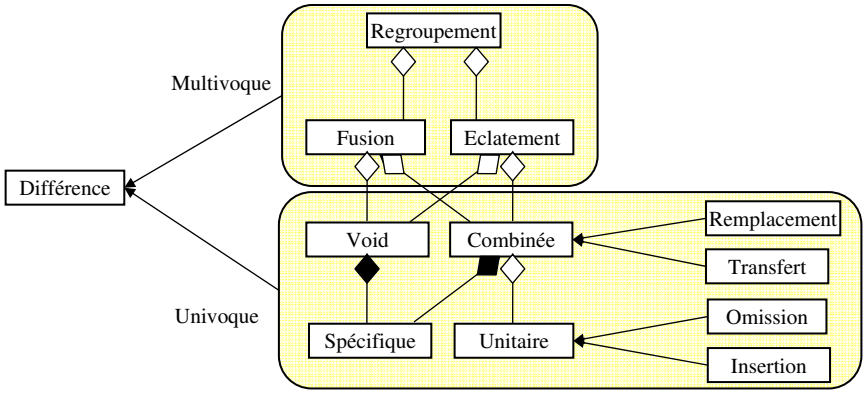
L'algorithme général de notre mesure de similarité de deux chaînes de caractères inclue un filtrage et une recherche des sous chaînes communes. Pour l'extraction des sous chaines communes, il s'inspire du MCWPA (***M**oving **C**ontracting **W**indow **P**attern **A**lgorithm*) [Yang *et al.* 2002]. Nous avons expérimenté plusieurs formules de calcul pour évaluer la similarité de deux chaines de caractères dont le calcul du SSNC (***S**um of the **S**quare of the **N**umber of the same **C**haracters*) et une formule en découlant contrainte par un seuil. L'algorithme en pseudo-code est le suivant :

```
Entrées : s et t, deux chaines de caractères
          seuil, score maximum de similarité à ne pas dépasser entre 2 chaînes de caractères
          windowsSizeDef, taille de la fenêtre minimum par défaut pour les sous chaînes
          filters, liste des filtres à appliquer sur les chaînes de caractères
Sorties : score, le score de similarité entre s et t tel que 0 ≤ score ≤ seuil
          differences, liste des différences entre s et t suite à l'application de filtres
Début
|  score ← 0
|  Si s xor t est vide Alors
|  |  // Calcul d'un score de similarité prenant en compte la longueur de la chaîne non vide
|  |  score ← minimum(seuil, (maximum(Length(s), Length(t)) + 1)²)
|  |  // avec SSNC : score ← (1.0 – 1 / (2 x maximum(Length(s), Length(t)))) x seuil
|  Sinon
|  |  Si s ≠ t Alors
|  |  |  // Clonage des chaînes de caractères pour les filtrer
|  |  |  sF ← clone(s) , tF ← clone(t)
|  |  |
|  |  |  // Filtres appliqués de manière itérative sur les chaînes de caractères sF et tF
|  |  |  differences ← filterStrings(sF,tF,filters)
|  |  |
|  |  |  Si sF ≠ tF Alors
|  |  |  |  // Longueur des chaînes sF et tF après les filtrages successifs
|  |  |  |  sFL ← Length(sF) , tFL ← Length(tF)
|  |  |  |
|  |  |  |  // Extraction des sous chaines communes de sF et tF inspirée du MCWPA
|  |  |  |  // Le troisième paramètre est la taille minimum de la fenêtre
|  |  |  |  commonSubStrings ← extractCommonSubStrings(sF, tF, minimum(windowsSizeDef, sFL, tFL))
|  |  |  |
|  |  |  |  // Calcul de la longueur totale des sous chaînes communes de sF et tF
|  |  |  |  subStringsL ← 0
|  |  |  |  Pour chaque subString de commonSubStrings Faire
|  |  |  |  |  subStringsL ← subStringsL + Length(subString)
|  |  |  |  |  // avec SSNC : subStringsL ← subStringsL + (2 x Length(subString))²)
|  |  |  |  Fin Pour
|  |  |  |
|  |  |  |  // Calcul du score de similarité
|  |  |  |  score ← minimum (seuil, (sFL + 1 – subStringsL) x (tFL + 1 - subStringL))
|  |  |  |  // avec SSNC : (1.0 - Math.sqrt((subStringsL) / (sFL+tFL)² ) ) x seuil
|  |  |  Fin Si
|  |  Fin Si
|  Fin Si
Fin
```

# Annexe 7 : Liste détaillée des différences relevées par ACDC

Dans cette annexe, nous détaillons l'ensemble des différences définies dans la taxonomie des différences (cf. partie 5.6) par le système ACDC.

Le schéma suivant montre comment les différences sont structurées les unes par rapport aux autres : par exemple, une différence de regroupement est constituée de plusieurs différences de fusion et d'éclatement qui sont elles-mêmes constituées de plusieurs différences combinées ou « Void », … Les différences de regroupement masquent donc les différences de fusion et d'éclatement qui masquent-elles mêmes les différences combinées qui les composent.




Nous présentons les différences des plus complexes aux plus simples, c'est-à-dire dans un premier temps les différences multivoques puis les différences univoques. Nous donnons un ou plusieurs exemples pour illustrer les différences. Ces exemples sont en majorité tirés du corpus d'exercices collectés lors des différentes expérimentations de l'environnement Diagram. Dans les différents exemples, le diagramme source est reporté à gauche et le diagramme cible est reporté à droite. Les motifs appartenant au diagramme source sont notés par la lettre *S* et ceux du diagramme cible sont notés par la lettre *T* (la première lettre de *Target* en anglais). Les éléments colorés en rouge dans le diagramme source sont des éléments insérés au plus bas niveau de granularité de notre taxonomie (i.e. différence unitaire d'insertion). Les éléments colorés en vert dans le diagramme cible sont des éléments omis au plus bas niveau de granularité de notre taxonomie (i.e. différence unitaire d'omission).

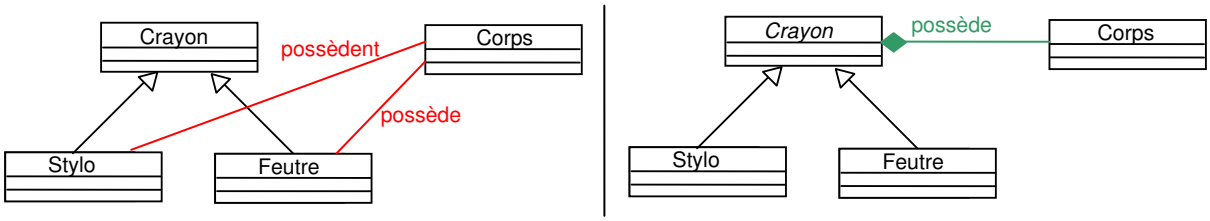
## *1. Les différences multivoques*

### 1.1 L'éclatement

{$S_1$ … $S_n$} ECLATEMENT {T} (*SPLIT*)

Le motif T est éclaté sous forme de n motifs $S_1$ … $S_n$. Les n motifs $S_1$ … $S_n$ correspondent au motif T.

*Exemple d'éclatement d'une relation d'association en deux relations d'association dans l'exercice « Stylo et Feutre » :*



*Trace des correspondances identifiées par l'ACDC :*

```
{<Association>possèdent(Stylo---Corps) <Association>possède(Feutre---Corps)} ECLATEMENT EQUIVALENT
{<Association>possède(Corps---Crayon)}
| {<Association>possèdent(Stylo---Corps)} TRANSFERT VERS_FILS {<Association>possède(Corps---Crayon)}
| | {<Association>possèdent(Stylo---Corps)} ASSOCIATION_EN_COMPOSITION {<Association>possède(Corps---Crayon)}
| | OMISSION {<Association>possède(Corps---Crayon)}
| | {<Association>possèdent(Stylo---Corps)} INSERTION
| {<Association>possède(Feutre---Corps)} TRANSFERT VERS_FILS {<Association>possède(Corps---Crayon)}
| | {<Association>possède(Feutre---Corps)} ASSOCIATION_EN_COMPOSITION {<Association>possède(Corps---Crayon)}
| | OMISSION {<Association>possède(Corps---Crayon)}
| | {<Association>possède(Feutre---Corps)} INSERTION
{<Class>Corps} VOID {<Class>Corps}
{<Class>Stylo} VOID {<Class>Stylo}
{<Class>Feutre} VOID {<Class>Feutre}
{<Class>Crayon} VOID {<Class>Crayon}
| {<Class>Crayon} NON_ABSTRAIT_EN_ABSTRAIT {<Class>Crayon}
{<Generalization>(Crayon<--Feutre)} VOID {<Generalization>(Crayon<--Feutre)}
{<Generalization>(Crayon<--Stylo)} VOID {<Generalization>(Crayon<--Stylo)}
```

## 1.2 La fusion

{S} FUSION {$T_1$ … $T_n$} (*MERGE*)

Le motif S est une fusion des n motifs $T_1$ … $T_n$. Les n motifs $T_1$ … $T_n$ correspondent au motif S.

*Exemple de fusion de deux attributs en un attribut dans l'exercice « Stylo et Feutre » :*


*Trace des correspondances identifiées par l'ACDC :*

```
{Crayon::<Property>couleur,marque} FUSION COMPLEMENTAIRE {Crayon::<Property>couleur Crayon::<Property>marque}
| {Crayon::<Property>couleur,marque} VOID {Crayon::<Property>couleur}
| {Crayon::<Property>couleur,marque} VOID {Crayon::<Property>marque}
{<Class>Crayon} VOID {<Class>Crayon}
```

*Exemple de fusion de deux associations en une association dans l'exercice « Stylo et Feutre » :*


*Trace des correspondances identifiées par l'ACDC (avec la fusion) :*

```
{<Association>utilisé(personne---Crayon)} FUSION {<Association>est utilisé par(Personne---Crayon)
<Association>appartient à(Personne---Crayon)}
| {<Association>utilisé(personne---Crayon)} VOID {<Association>est utilisé par(Personne---Crayon)}
| {<Association>utilisé(personne---Crayon)} VOID {<Association>appartient à(Personne---Crayon)}
{<Class>personne} VOID {<Class>Personne}
{<Class>Crayon} VOID {<Class>Crayon}
| {<Class>Crayon} NON_ABSTRAIT_EN_ABSTRAIT {<Class>Crayon}
```

*Trace des correspondances identifiées par l'ACDC (interprétation alternative sans la fusion) :*

```
{<Association>utilisé(personne---Crayon)} VOID {<Association>est utilisé par(Personne---Crayon)}
{<Class>personne} VOID {<Class>Personne}
{<Class>Crayon} VOID {<Class>Crayon}
| {<Class>Crayon} NON_ABSTRAIT_EN_ABSTRAIT {<Class>Crayon}
OMISSION {<Association>appartient à(Personne --- Crayon)}
```

## 1.3 Les mots clés pouvant préciser les fusions et les éclatements

{$S_1$ … $S_m$} EQUIVALENT {T} (*EQUIVALENT*)

Chacun des m motifs $S_1$ … $S_m$ correspondent individuellement au motif T ($S_1 \approx T$ et $S_2 \approx T$ et ... et $S_m \approx T$). Les motifs $S_1$ … $S_m$ sont redondants.

{$S_1$ … $S_m$} COMPLEMENTAIRE {T} (*SUPPLEMENT*)

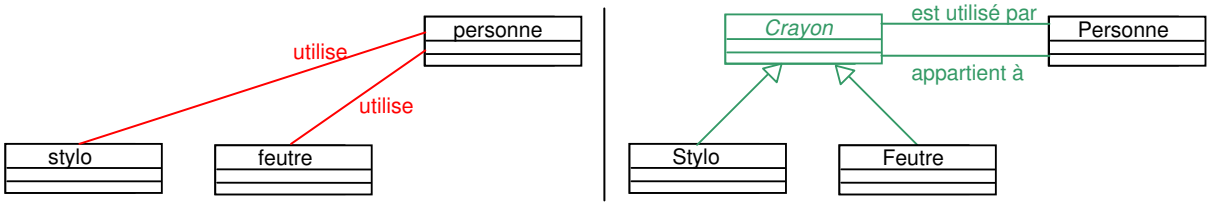
L'ensemble des motifs $S_1$ … $S_m$ correspondent au motif T ($S_1 + S_2 + \ldots + S_m \approx T$). Chacun des motifs S représente un fragment du motif T.

## 1.4 Le regroupement

{$S_1$ … $S_m$} REGROUPEMENT {$T_1$ … $T_n$} (*CLUSTER*)

Les m motifs $S_1$ … $S_m$ correspondent aux n motifs $T_1$ … $T_n$. Une différence de regroupement est constituée d'au moins une différence de fusion et d'une différence d'éclatement.

*Exemple de regroupement de deux associations en deux autres associations dans l'exercice « Stylo et Feutre » :*




*Trace des correspondances identifiées par l'ACDC (avec le regroupement) :*

```
{<Association>utilise(personne---stylo) <Association>utilise(personne---feutre)} REGROUPEMENT {<Association>est utilisé par(Personne---Crayon) <Association>appartient à(Personne---Crayon)}
| {<Association>utilise(personne---stylo) <Association>utilise(personne---feutre)} ECLATEMENT {<Association>est utilisé par(Personne---Crayon)}
| | {<Association>utilise(personne----stylo)} TRANSFERT VERS_FILS {Association>est utilisé par(Personne---Crayon)}
| | | OMISSION {Association>est utilisé par(Personne---Crayon)}
| | | {<Association>utilise(personne---stylo)} INSERTION
| | {<Association>utilise(personne---feutre)} TRANSFERT VERS_FILS {Association>est utilisé par(Personne---Crayon)}
| | | OMISSION {Association>est utilisé par(Personne---Crayon)}
| | | {<Association>utilise(personne---feutre)} INSERTION
| {<Association>utilise(personne---feutre)} FUSION {<Association>est utilisé par(Personne---Crayon) <Association>appartient à(Personne---Crayon)}
| | {<Association>utilise(personne---feutre)} TRANSFERT VERS_FILS {Association>est utilisé par(Personne---Crayon)}
| | | OMISSION {Association>est utilisé par(Personne---Crayon)}
| | | {<Association>utilise(personne---feutre)} INSERTION
| | {<Association>utilise(personne---feutre)} TRANSFERT VERS_FILS {<Association>appartient à(Personne---Crayon)}
| | | OMISSION {<Association>appartient à(Personne---Crayon)}
| | | {<Association>utilise(personne---feutre)} INSERTION
{<Class>personne} VOID {<Class>Personne}
{<Class>stylo} VOID {<Class>Stylo}
{<Class>feutre} VOID {<Class>Feutre}
OMISSION {<Class>Crayon}
OMISSION {<Generalization>(Crayon<--Feutre)}
OMISSION {<Generalization>(Crayon<--Stylo)}
```

# 2. Les différences univoques

## 2.1 Les différences univoques générales

### 2.1.1 Les différences générales unitaires

#### 2.1.1.1 L'insertion

{S} INSERTION (*INSERTION*)

Le motif S est présenté dans le diagramme source. Le motif S est absent du diagramme cible.

*Exemple d'insertion d'un attribut dans une classe dans l'exercice « Stylo et Feutre » :*


*Trace des correspondances identifiées par l'ACDC:*

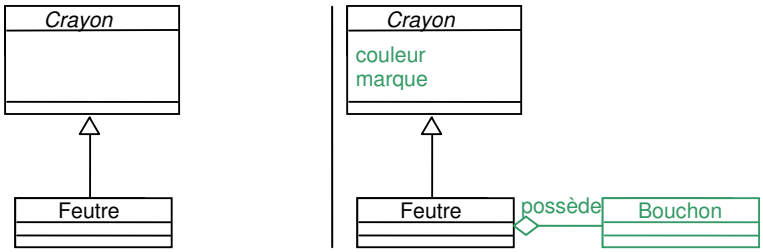
```
{<Class>Crayon} VOID {<Class>Crayon}
{Crayon::<Property>couleur} VOID {Crayon::<Property>couleur}
{Crayon::<Property>marque} VOID {Crayon::<Property>marque}
{Crayon::<Property>etc} INSERTION
```

#### 2.1.1.2 L'omission

OMISSION {T} (*OMISSION*)

Le motif T est absent du diagramme source. Le motif T est présent dans le diagramme cible.

*Exemple d'omission de plusieurs éléments dans l'exercice « Stylo et Feutre » :*




*Trace des correspondances identifiées par l'ACDC :*

```
{<Class>Crayon} VOID {<Class>Crayon}
{<Class>Feutre} VOID {<Class>Feutre}
OMISSION {<Class>Bouchon}
OMISSION {<Association>possède(Bouchon---Feutre)}
OMISSION {Crayon::<Property>couleur}
OMISSION {Crayon::<Property>marque}
```

### 2.1.2 Le void

{S} VOID {Y} (*VOID*)

Le motif S est apparié au motif T sans différence de structure. Le « void » n'est donc pas à proprement parler une différence mais un mot clé permettant de qualifier une correspondance univoque « sans différence de structure ».

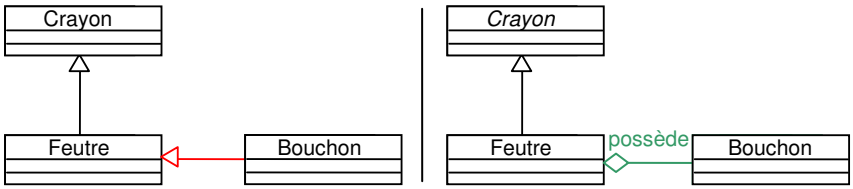
### *2.1.3 Les différences générales combinées*

#### 2.1.3.1 Le remplacement

{S} REMPLACEMENT {T} (*REPLACEMENT*)

Le motif S du diagramme source remplace le motif T du diagramme cible. Le motif S est inséré dans le diagramme source mais n'est pas présent dans le diagramme cible. Le motif T est omis dans le diagramme source mais est présent dans le diagramme cible.

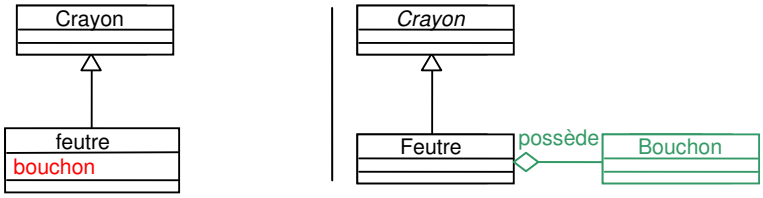
*Exemple de remplacement d'une relation d'association par une relation d'héritage dans l'exercice « Stylo et Feutre » :*




*Trace des correspondances identifiées par l'ACDC :*

```
{<Class>Crayon} VOID {<Class>Crayon}
| {<Class>Crayon} NON_ABSTRAIT_EN_ABSTRAIT {<Class>Crayon}
{<Class>Feutre} VOID {<Class>Feutre}
{<Class>Bouchon} VOID {<Class>Bouchon}
{<Generalization>(Crayon<--Feutre)} VOID {<Generalization>(Crayon<--Feutre)}
{<Generalization>(Feutre<--Bouchon)} REMPLACEMENT {<Association>possède(Bouchon---Feutre)}
| {<Generalization>(Feutre<--Bouchon)} NATURE_FRERE_DE {<Association>possède(Bouchon---Feutre)}
| OMISSION {<Association>possède(Bouchon---Feutre)}
| {<Generalization>(Feutre<--Bouchon)} INSERTION
```

*Exemple de remplacement d'une classe par un attribut dans l'exercice « Stylo et Feutre » :*




*Trace des correspondances identifiées par l'ACDC :*

```
{<Class>Crayon} VOID {<Class>Crayon}
| {<Class>Crayon} ABSTRACT {<Class>Crayon}
{<Class>feutre} VOID {<Class>Feutre}
{<Generalization>(Crayon<--feutre)} VOID {<Generalization>(Crayon<--Feutre)}
{feutre::<Property>bouchon} REMPLACEMENT {<Class>Bouchon}
| {feutre::<Property>bouchon} NATURE_INCOMPATIBLE {<Class>Bouchon}
| OMISSION {<Class>Bouchon}
| {feutre::<Property>Bouchon} INSERTION
OMISSION {<Association>possède(Bouchon---Feutre)}
```

#### 2.1.3.2 Le transfert

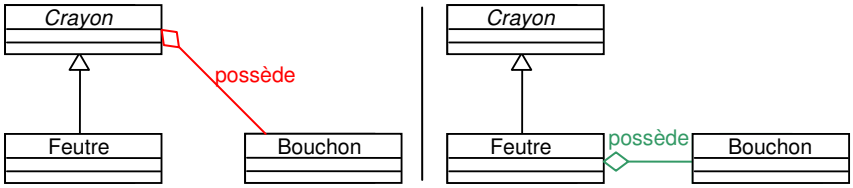
{S} TRANSFERT {T} (*TRANSFER*)

Le motif S du diagramme source correspondant au motif T dans le diagramme cible a été transféré à un endroit différent dans le diagramme source. Des mots clés qualifient plus précisément les différences de TRANSFERT dans le cas de transferts internes à des hiérarchies d'héritages (i.e. transferts liés à la propagation des propriétés et du comportement par héritage). Ces mots clés sont VERS_PERE, VERS_FILS et VERS_FRERE.

{S} VERS_PERE {T} (*UPPER*)

Le motif S est transféré plus « haut » dans l'arbre d'héritages contenant le motif T dans le diagramme cible.

*Exemple de transfert d'une relation vers une classe plus générale d'une hiérarchie d'héritages :*



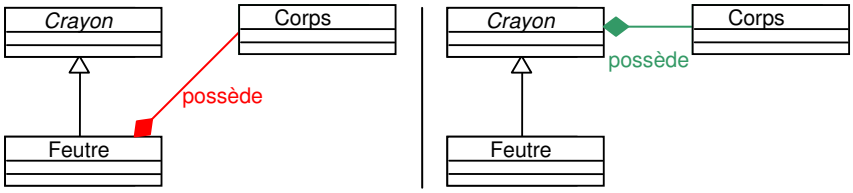

*Trace des correspondances identifiées par l'ACDC :*

```
{<Class>Crayon} VOID {<Class>Crayon}
{<Class>Feutre} VOID {<Class>Feutre}
{<Class>Bouchon} VOID {<Class>Bouchon}
{<Generalization>(Crayon<--Feutre)} VOID {<Generalization>(Crayon<--Feutre)}
{<Association>possède(Bouchon---Crayon)} TRANSFERT VERS_PERE {<Association>possède(Bouchon---Feutre)}
| OMISSION {<Association>possède(Bouchon---Feutre)}
| {<Association>possède(Bouchon---Crayon)} INSERTION
```

{S} VERS_FILS {T} (*LOWER*)

Le motif S est transféré plus « bas » dans l'arbre d'héritages contenant le motif T dans le diagramme cible.

*Exemple de transfert d'une relation vers une classe plus spécialisée d'une hiérarchie d'héritages :*



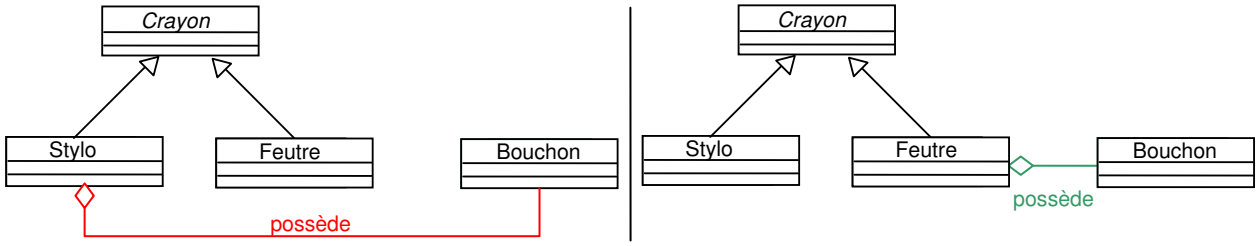

*Trace des correspondances identifiées par l'ACDC :*

```
{<Class>Crayon} VOID {<Class>Crayon}
{<Class>Feutre} VOID {<Class>Feutre}
{<Class>Corps} VOID {<Class>Corps}
{<Generalization>(Crayon<--Feutre)} VOID {<Generalization>(Crayon<--Feutre)}
{<Association>possède(Corps---Feutre)} TRANSFERT VERS_FILS {<Association>possède(Corps---Crayon)}
| OMISSION {<Association>possède(Corps---Feutre)}
| {<Association>possède(Corps---Crayon)} INSERTION
```

{S} VERS_FRERE {T} (*BROTHER*)

Le motif S du diagramme source est transféré vers un motif « frère » dans l'arbre d'héritages contenant le motif T dans le diagramme cible.

*Exemple de transfert d'une relation vers une classe sœur d'une hiérarchie d'héritages :*



*Trace des correspondances identifiées par l'ACDC :*

```
{<Class>Crayon} VOID {<Class>Crayon}
{<Class>Feutre} VOID {<Class>Feutre}
{<Class>Stylo} VOID {<Class>Stylo}
{<Class>Bouchon} VOID {<Class>Bouchon}
{<Generalization>(Crayon<--Feutre)} VOID {<Generalization>(Crayon<--Feutre)}
{<Generalization>(Crayon<--Stylo)} VOID {<Generalization>(Crayon<--Stylo)}
{<Association>possède(Bouchon---Stylo)} TRANSFERT VERS_FRERE {<Association>possède(Bouchon---Feutre)}
| OMISSION {<Association>possède(Bouchon---Feutre)}
| {<Association>possède(Bouchon---Stylo)} INSERTION
```

## 2.2 Les différences univoques spécifiques (aux motifs simples)

### *2.2.1 Les différences univoques spécifiques à la nature des éléments*

Les différences spécifiques à la nature des éléments UML sont identifiées à l'aide des classes UML et des interfaces UML des éléments UML (cf. Annexe 2).

{S} NATURE_HERITE_DE {T} (*NATURE_SPECIALIZES*)

La classe de l'élément T est une superclasse de la classe de l'élément S. La classe de S hérite de celle de T.

*Exemples :*

- Une classe ou une relation d'association du diagramme cible remplacée par une classe d'association dans le diagramme source (classe commune : Classe ou Association).

{S} NATURE_HERITE_PAR {T} (*NATURE_GENERALIZES*)

La classe de l'élément S est une superclasse de la classe de l'élément T. La classe de T hérite de la classe de S.

*Exemples :*

- Une classe d'association du diagramme cible remplacée par une classe ou une relation d'association du diagramme source (classe commune : Classe ou Association).

{S} NATURE_FRERE_DE {T} (*NATURE_BROTHERS*)

Les classes des éléments S et T héritent d'une classe ou implémentent une interface commune.

*Exemples :*

- Une classe remplacée par une interface, un objet, une association ou n'importe quelle autre combinaison de ces 4 types d'éléments dans le diagramme source (classe commune : Classificateur).
- Une relation d'héritage du diagramme remplacée par une relation de dépendance ou une relation d'association (association simple, agrégation ou composition) ou n'importe quelle autre combinaison de ces 3 types d'éléments (classe commune : Relation).
- Une propriété (attribut ou fin d'association) remplacée par une opération ou inversement (classe commune : Caractéristique).

{S} NATURE_INCOMPATIBLE {T} (*NATURE_INCOMPATIBLE*)

Les classes des éléments S et T n'ont pas de superclasse ou d'interface commune.

*Exemples :*

- Un classificateur (hors relation d'association) remplacée par une relation ou une caractéristique ou n'importe quelle autre combinaison de ces 3 types d'éléments.

### 2.2.2 Les différences spécifiques à la visibilité des éléments

Ces différences ne sont pas utilisées dans l'environnement Diagram car les visibilités ne sont pas représentables dans les diagrammes de classes d'analyse élaborée dans ce dernier. Les différences de visibilité possibles sont les suivantes :

| | | | |
|---|---|---|---|
| {S} | PROTEGE_EN_PRIVE | {T} | (*PROTECTED_TO_PRIVATE*) |
| {S} | PUBLIC_EN_PRIVE | {T} | (*PUBLIC_TO_PRIVATE*) |
| {S} | PAQUETAGE_EN_PRIVE | {T} | (*PACKAGE_TO_PRIVATE*) |
| {S} | PRIVE_EN_PROTEGE | {T} | (*PRIVATE_TO_PROTECTED*) |
| {S} | PUBLIC_EN_PROTEGE | {T} | (*PUBLIC_TO_PROTECTED*) |
| {S} | PAQUETAGE_EN_PROTEGE | {T} | (*PACKAGE_TO_PROTECTED*) |
| {S} | PROTEGE_EN_PUBLIC | {T} | (*PROTECTED_TO_PUBLIC*) |
| {S} | PRIVEE_EN_PUBLIC | {T} | (*PRIVATE_TO_PUBLIC*) |
| {S} | PAQUETAGE_EN_PUBLIC | {T} | (*PACKAGE_TO_PUBLIC*) |
| {S} | PROTEGE_EN_PAQUETAGE | {T} | (PROTECTED_*TO_PACKAGE*) |
| {S} | PUBLIC_EN_PAQUETAGE | {T} | (*PUBLIC_TO_PACKAGE*) |
| {S} | PRIVE_EN_PAQUETAGE | {T} | (*PRIVATE_TO_PACKAGE*) |

### 2.2.3 Les différences spécifiques au fait que des éléments sont isolés ou non

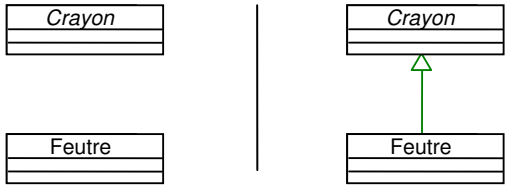
| | | | |
|---|---|---|---|
| ({S}) | NON_ISOLE_EN_ISOLE | {T} | (NOT_*ORPHANED_TO_ORPHANED*) |

L'élément S n'est pas isolé alors que l'élément T l'est.

| | | | |
|---|---|---|---|
| {S} | ISOLE_EN_NON_ISOLE | ({T}) | (*ORPHANED_TO_NOT_ORPHANED*) |

L'élément S est orphelin alors que l'élément T ne l'est pas.

*Exemple de classe isolée :*




*Trace des correspondances identifiées par l'ACDC :*

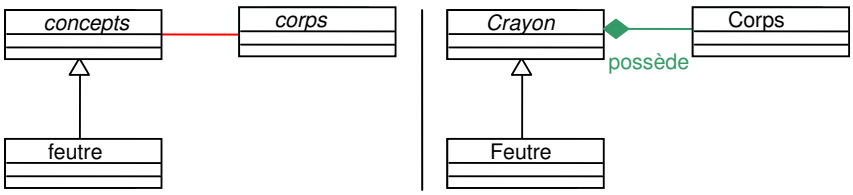
```
{<Class>Crayon} VOID {<Class>Crayon}
| {<Class>Crayon} ISOLE_EN_NON_ISOLE {<Class>Crayon}
{<Class>Feutre} VOID {<Class>Feutre}
OMISSION {<Generalization>(Crayon<--Feutre)}
```

### 2.2.4 Les différences spécifiques à la présence ou l'absence du nom d'un élément

| | | | |
|---|---|---|---|
| {S} | NON_NOMME_EN_NOMME | {T} | (*NOT_NAMED_TO_NAMED*) |

L'élément S n'est pas nommé alors que l'élément T est nommé.

*Exemple de relation d'association non nommées correspondant à une relation d'association nommée :*



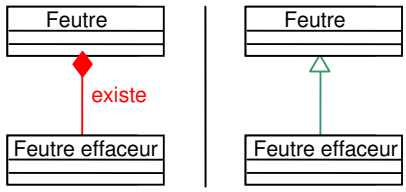

*Trace des correspondances identifiées par l'ACDC :*

```
{<Association>(corps---concepts)} VOID {<Association>possède(Corps---Crayon)}
| {<Association>(corps---concepts)} NON_NOMME_EN_NOMME ASSOCIATION_EN_COMPOSITION {<Association>possède(Corps---Crayon)}
{<Class>concepts} VOID {<Class>Crayon}
{<Class>corps} VOID {<Class>Corps}
| {<Class>corps} ABSTRAIT_EN_NON_ABSTRAIT {<Class>Corps}
{<Class>feutre} VOID {<Class>Feutre}
```

{S} NOMME_EN_NON_NOMME {T} (*NAMED_TO_NOT_NAMED*)

L'élément S est nommé alors que l'élément T n'est pas nommé.

*Exemple de relation d'association nommée correspondant à une relation d'héritage :*




*Trace des correspondances identifiées par l'ACDC :*

```
{<Association>existe(Feutre effaceur---Feutre)} REMPLACEMENT {<Generalization>(Feutre<--Feutre effaceur)}
| {<Association>existe(Feutre effaceur---Feutre)} NATURE_FRERE_DE NOMME_EN_NON_NOMME {<Generalization>(Feutre<--Feutre effaceur)}
| OMISSION {<Generalization>(Feutre<--Feutre effaceur)}
| {<Association>existe(Feutre effaceur---Feutre)} INSERTION
{<Class>Feutre effaceur} VOID {<Class>Feutre effaceur}
{<Class>Feutre} VOID {<Class>Feutre}
```

### *2.2.5 Les différences spécifiques aux chaînes de caractères des éléments nommés*

Ces différences concernent les transformations appliquées sur les chaînes de caractères lors de leur comparaison une à une.

{S} SEPARATEURS_IGNORES {T} (*IGNORE_SEPARATORS*)

Les séparateurs (espaces, tabulations, retours à la ligne...) ont été supprimés lors de la comparaison des chaînes S et T.

{S} ACCENTS_IGNORES {T} (*IGNORE_ACENTS*)

Les caractères accentués (é, è, â, ï...) ont été transformés en caractères non accentués lors de la comparaison des chaînes S et T.

{S} CARACTERES_SPECIAUX_IGNORES {T} (*IGNORE_SPECIAL_CHARACTERS*)

Les caractères spéciaux (!, :, ; ;, -...) ont été supprimés lors de la comparaison des chaînes S et T.

{S} CASSE_IGNOREE {T} (*IGNORE_CASE*)

Les caractères en majuscules ont été transformés en caractères en minuscules lors de la comparaison des chaînes S et T.

{S} NOMBRE_IGNORE {T} (*IGNORE_NUMBER*)

Les mots au pluriel ont été mis au singulier lors de la comparaison des chaînes S et T.

{S} GENRE_IGNORE {T} (*IGNORE_GENDER*)

Les mots au féminin ont été mis au masculin lors de la comparaison des chaînes S et T.

### *2.2.6 Les différences spécifiques à l'abstraction des classificateurs*

{S} NON_ABSTRAIT_EN_ABSTRAIT {T} (*NOT_ABSTRACT_TO_ABSTRACT*)

L'élément S n'est pas abstrait alors que l'élément T est abstrait.

{S} ABSTRAIT_EN_NON_ABSTRAIT {T} (*ABSTRACT_TO_NOT_ABSTRACT*)

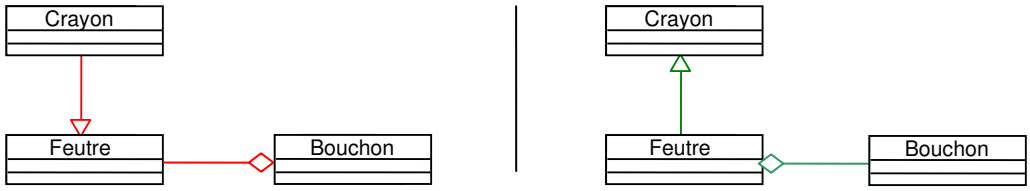
L'élément S est abstrait alors que l'élément T n'est pas abstrait.

### *2.2.7 Les différences spécifiques aux sens et à la complétude des relations*

{S} INVERSE {T} (*REVERSE*)

Le sens de la relation de S est inversé par rapport au sens de la relation de T.

*Exemple de deux relations sont le sens d'orientation est inversé :*




*Trace des correspondances identifiées par l'ACDC :*

```
{<Class>Crayon} VOID {<Class>Crayon}
{<Class>Feutre} VOID {<Class>Feutre}
{<Class>Bouchon} VOID {<Class>Bouchon}
{<Association>(Bouchon---Feutre)} VOID {<Association>(Feutre---Bouchon)}
| {<Association>(Bouchon---Feutre)} INVERSE {<Association>(Feutre---Bouchon)}
{<Generalization>(Feutre<--Crayon)} VOID {<Generalization>(Crayon<--Feutre)}
| {<Generalization>(Feutre<--Crayon)} INVERSE {<Generalization>(Crayon<--Feutre)}
```

{S} INCOMPLET ({T}) (*INCOMPLETE*)

La relation S est incomplète. Elle n'est liée qu'à 0 ou 1 élément. Une erreur syntaxique de type 3.2 est associée à cette différence (cf. figure 53 du chapitre 6).

### *2.2.8 Les différences spécifiques à la dérivation des associations et des propriétés*

Ces différences ne sont pas utilisées dans l'environnement Diagram car la représentation d'éléments dérivés n'est pas possibles dans les diagrammes de classes d'analyse élaborés dans ce dernier.

{S} NON_DERIVE_EN_DERIVE {T} (*NOT_DERIVED_TO_DERIVED*)

L'élément S n'est pas dérivé alors que l'élément T est dérivé.

{S} DERIVE_EN_NON_DERIVE {T} (*DERIVED_TO_NOT_DERIVED*)

L'élément S est dérivé alors que l'élément T n'est pas dérivé.

### *2.2.9 Les différences spécifiques au type d'agrégation des relations d'association et des propriétés*

#### 2.2.9.1 Différence de type d'agrégation avec perte d'information

{S} AGREGATION_EN_COMPOSITION {T} (*AGGREGATION_TO_COMPOSITION*)
La relation S est une agrégation alors que la relation T est une composition.
{S} AGREGAT_A_COMPOSITE {T} (*SHARED_TO_COMPOSITE*)
La fin d'association S est de type agrégat alors que la fin d'association T est de type composite.

{S} ASSOCIATION_EN_AGREGATION {T} (ASSOCIATION_*TO_AGGREGATION*)
La relation S est une association alors que la relation T est une agrégation.
{S} AUCUN_A_AGREGAT {T} (*NONE_TO_SHARED*)
La fin d'association S est de type aucun alors que la fin d'association T est de type agrégat.

{S} ASSOCIATION_EN_COMPOSITION {T} (ASSOCIATION_*TO_COMPOSITION*)
La relation S est une association alors que la relation T est une composition.
{S} AUCUN_A_COMPOSITE {T} (*NONE_TO_COMPOSITE*)
La fin d'association S est de type aucun alors que la fin d'association T est de type composite.

#### 2.2.9.2 Différence de type d'agrégation avec ajout d'information

{S} AGREGATION_EN_ASSOCIATION {T} (*AGGREGATION_TO_ASSOCIATION*)
La relation S est une agrégation alors que la relation T est une association.
{S} AGREGAT_A_AUCUN {T} (*SHARED_TO_NONE*)
La fin d'association S est de type agrégat alors que la fin d'association T est de type aucun.

{S} COMPOSITION_EN_AGREGATION {T} (*COMPOSITION_TO_AGGREGATION*)
La relation S est une composition alors que la relation T est une agrégation.
{S} COMPOSITE_A_AGREGAT {T} (*COMPOSITE_TO_SHARED*)
La fin d'association S est de type composite alors que la fin d'association T est de type agrégat.

{S} COMPOSITION_EN_ASSOCIATION {T} (*COMPOSITION_TO_ASSOCIATION*)
La relation S est une composition alors que la relation T est une association.
{S} COMPOSITE_A_AUCUN {T} (*COMPOSITE_TO_NONE*)
La fin d'association S est de type composite alors que la fin d'association T est de type aucun.

#### 2.2.9.3 Différence de type d'association incorrect

{S} MAUVAIS_TYPE_D_ASSOCIATION ({T}) (*BAD_ASSOCIATION_KIND*)
La relation S est incorrecte au niveau de son type d'agrégation. Cette différence est complétée par une erreur syntaxique de type 3.2 (cf. figure 53 du chapitre 6).

*Exemples :*

- Deux types d'agrégation « composition » ou « agrégation » au niveau des deux fins d'association d'une relation d'association binaire.
- Un type d'agrégation « composition » ou « agrégation » au niveau d'une ou plusieurs fins d'association d'une relation d'association n-aire.

### *2.2.10 Les différences spécifiques aux multiplicités des propriétés*

Les différences concernant les multiplicités des propriétés sont caractérisables au niveau des multiplicités basses et hautes individuellement et au niveau de l'intervalle complet de multiplicités. Nous nous sommes inspirés des intervalles temporels d'Allen [Allen 1983] pour définir les différences d'intervalles de multiplicités.

#### 2.2.10.1 Différences par rapport aux multiplicités basses des propriétés S et T

{S} MULTIPLICITE_BASSE_AVANT_BASSE {T} (*MULTIPLICITY_LOWER_BEFORE_LOWER*)
La multiplicité basse de la propriété S est strictement inférieure à la multiplicité basse de la propriété T.

{S} MULTIPLICITE_BASSE_APRES_BASSE {T} (*MULTIPLICITY_LOWER_AFTER_LOWER*)
La multiplicité basse de la propriété S est strictement supérieure à la multiplicité inférieure de la propriété T.

{S} MULTIPLICITE_BASSE_EGALE_BASSE {T} (*MULTIPLICITY_LOWER_EQUALS_LOWER*)
La multiplicité basse de la propriété S est égale à la multiplicité basse de la propriété T.

#### 2.2.10.2 Différences par rapport aux multiplicités hautes des propriétés S et T

{S} MULTIPLICITE_HAUTE_AVANT_HAUTE {T} (*MULTIPLICITY_UPPER_BEFORE_UPPER*)
La multiplicité haute de la propriété S est strictement inférieure à la multiplicité haute de la propriété T.

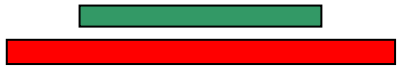
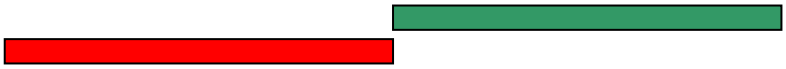
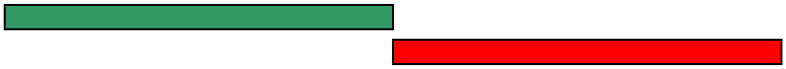
{S} MULTIPLICITE_HAUTE_APRES_HAUTE {T} (*MULTIPLICITY_UPPER_AFTER_UPPER*)
La multiplicité haute de la propriété S est strictement supérieure à la multiplicité haute de la propriété T.

{S} MULTIPLICITE_HAUTE_EGALE_HAUTE {T} (*MULTIPLICITY_UPPER_EQUALS_UPPER*)
La multiplicité haute de la propriété S est égale à la multiplicité haute de la propriété T.

#### 2.2.10.3 Différences par rapport aux intervalles de multiplicités (intervalles temporels d'Allen)

| | | | |
|---|---|---|---|
| {S} | MULTIPLICITES_EGALES | {T} | (*MULTIPLICITIES_EQUALS*) |
| {S} | MULTIPLICITES_AVANT | {T} | (*MULTIPLICITIES_BEFORE*) |
| {S} | MULTIPLICITES_APRES | {T} | (*MULTIPLICITIES_AFTER*) |
| {S} | MULTIPLICITES_PENDANT | {T} | (*MULTIPLICITIES_DURING*) |
| {S} | MULTIPLICITES_CONTIENNENT | {T} | (*MULTIPLICITIES_CONTAIN*) |
| {S} | MULTIPLICITES_CHEVAUCHENT | {T} | (*MULTIPLICITIES_OVERLAP*) |
| {S} | MULTIPLICITES_CHEVAUCHEES_PAR | {T} | (*MULTIPLICITIES_OVERLAPED_BY*) |
| {S} | MULTIPLICITES_RENCONTRENT | {T} | (*MULTIPLICITIES_MEET*) |
| {S} | MULTIPLICITES_RENCONTREES_PAR | {T} | (*MULTIPLICITIES_MET_BY*) |
| {S} | MULTIPLICITES_COMMENCENT | {T} | (*MULTIPLICITIES_START*) |
| {S} | MULTIPLICITES_COMMENCEES_PAR | {T} | (*MULTIPLICITIES_STARTED_BY*) |
| {S} | MULTIPLICITES_TERMINENT | {T} | (*MULTIPLICITIES_FINISH*) |
| {S} | MULTIPLICITES_TERMINEES_PAR | {T} | (*MULTIPLICITIES_FINISHED_BY*) |

Les différences concernant les multiplicités sous forme d'intervalles sont illustrées dans le tableau suivant. Les rectangles verts représentent les intervalles de multiplicités des propriétés sources et les rectangles en rouge représentent les multiplicités des propriétés cibles.

| Relations | Sémantique graphique |
|---|---|
| S equals T |  |
| S before T |  |
| S after T |  |
| S during T |  |
| S contains T |  |
| S overlaps T |  |
| S overlaped by T |  |
| S meets T |  |
| S met by T |  |
| S starts T |  |
| S started by T |  |
| S finishes T |  |
| S finished by T |  |

**2.2.10.4 Différence de formatage d'une multiplicité**

{S} MAUVAIS_FORMAT_DES_MULTIPLICITES {T} (*MULTIPLICITIES_HAS_BAD_FORMAT*)

La multiplicité de la propriété S a un format incorrect (borne inférieure plus grande que la borne supérieure, absence de l'une des bornes, borne négative…). Cette différence est complétée par des erreurs syntaxiques de types 4.4 à 4.11 en fonction des problèmes de format (cf. figure 53 du chapitre 6).

### *2.2.11 Exemple illustrant les différences de type d'agrégation, de multiplicités, d'inversion de sens des associations et de leurs propriétés associées*

*Diagramme source :*

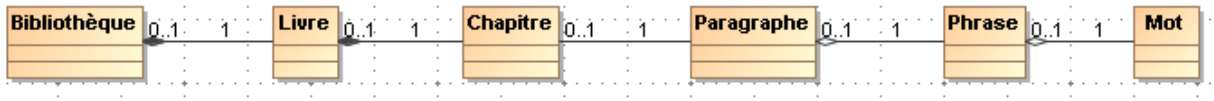
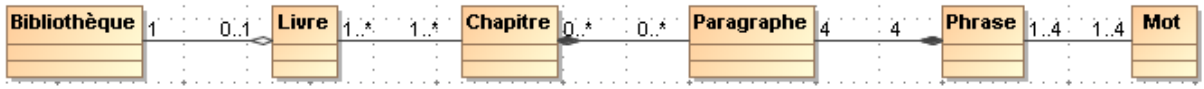



*Diagramme cible :*




*Trace des correspondances identifiées par l'ACDC :*

Les traces de cet exemple se lisent directement en parcourant les diagrammes de gauche à droite.

Attention, le sens de lecture des propriétés est inversée par rapport à la logique de lecture des multiplicités car les multiplicités d'un coté d'une association (la classe de gauche par exemple) sont stockées dans la fin d'association de l'autre coté (la classe de droite dans notre exemple) et réciproquement.

```
{<Class>Bibliothèque} VOID {<Class>Bibliothèque}
{<Class>Livre} VOID {<Class>Livre}
{<Class>Chapitre} VOID {<Class>Chapitre}
{<Class>Paragraphe} VOID {<Class>Paragraphe}
{<Class>Phrase} VOID {<Class>Phrase}
{<Class>Mot} VOID {<Class>Mot}

{<Association>(Livre---Bibliothèque)} VOID {<Association>(Bibliothèque---Livre)}
| {<Association>(Livre---Bibliothèque)} COMPOSITION_EN_AGREGATION INVERSE {<Association>(Bibliothèque---Livre)}
| {Livre(Livre---Bibliothèque)::<Property>} AUCUN_A_AGREGAT  MULTIPLICITES_TERMINEES_PAR {Livre(Bibliothèque---
Livre)::<Property>}
| {Bibliothèque(Livre---Bibliothèque)::<Property>} COMPOSITE_A_AUCUN  MULTIPLICITES_TERMINENT
{Bibliothèque(Bibliothèque---Livre)::<Property>}

{<Association>(Chapitre---Livre)} VOID {<Association>(Livre---Chapitre)}
| {<Association>(Chapitre---Livre)} COMPOSITION_EN_ASSOCIATION {<Association>(Livre---Chapitre)}
| {Chapitre(Chapitre---Livre)::<Property>} MULTIPLICITES_RENCONTRENT {Chapitre(Livre---Chapitre)::<Property>}
| {Livre(Chapitre---Livre)::<Property>} COMPOSITE_A_AUCUN  MULTIPLICITES_COMMENCENT {Livre(Livre---
Chapitre)::<Property>}

{<Association>(Paragraphe----Chapitre)} VOID {<Association>(Chapitre---Paragraphe)}
| {<Association>(Paragraphe---Chapitre)} ASSOCIATION_EN_COMPOSITION {<Association>(Chapitre---Paragraphe)}
| {Paragraphe(Paragraphe---Chapitre)::<Property>} MULTIPLICITES_COMMENCENT {Paragraphe(Chapitre---
Paragraphe)::<Property>}
| {Chapitre(Paragraphe---Chapitre)::<Property>} AUCUN_A_COMPOSITE  MULTIPLICITES_PENDANT {Chapitre(Chapitre---
Paragraphe)::<Property>}

{<Association>(Phrase---Paragraphe)} VOID {<Association>(Paragraphe---Phrase)}
| {<Association>(Phrase---Paragraphe)} AGREGATION_EN_COMPOSITION  INVERSE {<Association>(Paragraphe---Phrase)}
| {Phrase(Phrase---Paragraphe)::<Property>} AUCUN_A_COMPOSITE  MULTIPLICITES_AVANT {Phrase(Paragraphe---
Phrase)::<Property>}
| {Paragraphe(Phrase---Paragraphe)::<Property>} AGREGAT_A_AUCUN  MULTIPLICITES_AVANT {Paragraphe(Paragraphe---
Phrase)::<Property>}

{<Association>(Mot---Phrase)} VOID {<Association>(Mot---Phrase)}
| {<Association>(Mot---Phrase)} AGREGATION_EN_ASSOCIATION {<Association>(Mot---Phrase)}
| {Mot(Mot---Phrase)::<Property>} MULTIPLICITES_RENCONTRENT {Mot(Mot---Phrase)::<Property>}
| {Phrase(Mot---Phrase)::<Property>} AGREGAT_A_AUCUN  MULTIPLICITES_COMMENCENT {Phrase(Mot---Phrase)::<Property>}
```

# Annexe 8 : Exemple de sorties textuelles du diagnostic produit par ACDC

Le diagnostic suivant a été réalisé sur les diagrammes présentés dans la figure 48 de la partie 5.6.3 pour l'exercice « Cirque » dont l'énoncé et le diagramme de référence sont reportés également dans la partie 7.2.2.2.

```
---------- Hiérarchie de correspondances qualifiées par des différences structurelles ----------
[apprenant.uml] --> [corrigé.uml] :  DIFFERENT 160 <- 270/2 + 51/2   107
|
| {<Class>chauffeurs de poids lourds <Class>chauffeurs} ECLATEMENT {<Class>Chauffeur de poids lourd}
| | {chauffeurs de poids lourds} CASSE_IGNOREE  SEPARATEURS_IGNORES {Chauffeur de poids lourd}
| | {chauffeurs} CASSE_IGNOREE {Chauffeur}
| | {<Class>chauffeurs de poids lourds} VOID {<Class>Chauffeur de poids lourd}
| | {<Class>chauffeurs} VOID {<Class>Chauffeur de poids lourd}
| {<Class>numéros} VOID {<Class>Numéro}
| | {numéros} CASSE_IGNOREE  ACCENTS_IGNORES {Numéro}
| {<Class>artistes} VOID {<Class>Artiste}
| | {artistes} CASSE_IGNOREE  NOMBRE_IGNORE {Artiste}
| {<Class>accessoires} VOID {<Class>Accessoire}
| | {accessoires} CASSE_IGNOREE {Accessoire}
| {<Class>dressage} VOID {<Class>Numéro de dressage}
| | {dressage} CASSE_IGNOREE  ACCENTS_IGNORES  SEPARATEURS_IGNORES {Numéro de dressage}
| {<Class>musicaux} VOID {<Class>Numéro musical}
| | {musicaux} CASSE_IGNOREE  NOMBRE_IGNORE  ACCENTS_IGNORES  SEPARATEURS_IGNORES {Numéro musical}
| {<Class>membres du personnel} VOID {<Class>Membre du personnel du cirque}
| | {membres du personnel} CASSE_IGNOREE  SEPARATEURS_IGNORES {Membre du personnel du cirque}
| {<Class>camions} VOID {<Class>Camion}
| | {camions} CASSE_IGNOREE {Camion}
| {<Association>(numéros---artistes)} VOID {<Association>assure(Artiste---Numéro)}
| | {<Association>(numéros---artistes)} NON_NOMME_EN_NOMME {<Association>assure(Artiste---Numéro)}
| {<Association>peut nécessiter(numéros---accessoires)} VOID {<Association>nécessite(Numéro---Accessoire)}
| | {peut nécessiter} ACCENTS_IGNORES  SEPARATEURS_IGNORES {nécessite}
| {<Generalization>(membres du personnel<--artistes)} VOID {<Generalization>(Membre du personnel du cirque<--
Artiste)}
| {<Generalization>(numéros<--dressage)} VOID {<Generalization>(Numéro<--Numéro de dressage)}
| {<Generalization>(numéros<--musicaux)} VOID {<Generalization>(Numéro<--Numéro musical)}
| {<Association>sont rangés(accessoires---camions)} VOID {<Association>est rangé dans(Camion---Accessoire)}
| | {sont rangés} NOMBRE_IGNORE  ACCENTS_IGNORES  SEPARATEURS_IGNORES {est rangé dans}
| {<Generalization>(membres du personnel<--chauffeurs de poids lourds)} VOID {<Generalization>(Membre du personnel
du cirque<--Chauffeur de poids lourd)}
| {<Association>est assigné(chauffeurs---camions)} VOID {<Association>est assigné(Camion---Chauffeur de poids
lourd)}
| {artistes::<Property>pseudonymes des artistes} VOID {Artiste::<Property>pseudonymes}
| | {pseudonymes des artistes} NOMBRE_IGNORE  SEPARATEURS_IGNORES {pseudonymes}
| {numéros::<Property>code} VOID {Numéro::<Property>code}
| {numéros::<Property>nom} VOID {Numéro::<Property>nom}
| {numéros::<Property>nombre d artistes} VOID {Numéro::<Property>nombre d artistes présents sur scène}
| | {nombre d artistes} NOMBRE_IGNORE  ACCENTS_IGNORES  SEPARATEURS_IGNORES {nombre d artistes présents sur scène}
| {numéros::<Property>durée} VOID {Numéro::<Property>durée}
| {accessoires::<Property>numéro de série} VOID {Accessoire::<Property>numéro de série}
| {accessoires::<Property>désignation} VOID {Accessoire::<Property>désignation}
| {accessoires::<Property>couleur} VOID {Accessoire::<Property>couleur}
| {accessoires::<Property>volume} VOID {Accessoire::<Property>volume}
| {membres du personnel::<Property>salaire} VOID {Membre du personnel du cirque::<Property>salaire}
| {membres du personnel::<Property>date de naissance} VOID {Membre du personnel du cirque::<Property>date de
naissance}
| {membres du personnel::<Property>prénom} VOID {Membre du personnel du cirque::<Property>prénom}
| {membres du personnel::<Property>nom} VOID {Membre du personnel du cirque::<Property>nom}
| {membres du personnel::<Property>numéro} VOID {Membre du personnel du cirque::<Property>numéro INSEE}
| | {numéro} CASSE_IGNOREE  GENRE_IGNORE  ACCENTS_IGNORES  SEPARATEURS_IGNORES {numéro INSEE}
| {camions::<Property>numéro d immatriculation} VOID {Camion::<Property>numéro d immatriculation}
| {camions::<Property>marque} VOID {Camion::<Property>marque}
| {camions::<Property>modèle} VOID {Camion::<Property>modèle}
| {camions::<Property>capacité} VOID {Camion::<Property>capacité}
| {chauffeurs de poids lourds::<Property>numéro du permis de conduire} VOID {Chauffeur de poids
lourd::<Property>numero du permis de conduire}
```

```
| | {numéro du permis de conduire} ACCENTS_IGNORES {numero du permis de conduire}
| {<Class>animal} REMPLACEMENT {Numéro de dressage::<Property>animal concerné}
| |   OMISSION {Numéro de dressage::<Property>animal concerné}
| | {<Class>animal} INSERTION
| | {<Class>animal} NATURE_INCOMPATIBLE  PRIVE_EN_PUBLIC {Numéro de dressage::<Property>animal concerné}
| {<Class>instrument} REMPLACEMENT {Numéro musical::<Property>instrument utilisé}
| |   OMISSION {Numéro musical::<Property>instrument utilisé}
| | {<Class>instrument} INSERTION
| | {<Class>instrument} NATURE_INCOMPATIBLE  PRIVE_EN_PUBLIC {Numéro musical::<Property>instrument utilisé}
|   OMISSION {<Class>Numéro d acrobatie}
|   OMISSION {<Generalization>(Numéro<--Numéro d acrobatie)}
|   OMISSION {Numéro d acrobatie::<Property>type}
|   OMISSION {<Association>utilisé(Artiste---Accessoire)}
| {<Association>concerné par(dressage---animal)} INSERTION
| {<Association>utilisé pour(musicaux---instrument)} INSERTION

---------- Correspondances qualifiées par type de différences et par type de motifs simples ----------
Différences multivoques des motifs simples:
{<Class>chauffeurs de poids lourds <Class>chauffeurs} ECLATEMENT {<Class>Chauffeur de poids lourd}

Différences univoques combinées des classes :
{<Class>chauffeurs de poids lourds} VOID {<Class>Chauffeur de poids lourd}
{<Class>membres du personnel} VOID {<Class>Membre du personnel du cirque}
{<Class>artistes} VOID {<Class>Artiste}
{<Class>numéros} VOID {<Class>Numéro}
{<Class>accessoires} VOID {<Class>Accessoire}
{<Class>camions} VOID {<Class>Camion}
{<Class>chauffeurs} VOID {<Class>Chauffeur de poids lourd}
{<Class>musicaux} VOID {<Class>Numéro musical}
{<Class>dressage} VOID {<Class>Numéro de dressage}
{<Class>animal} REMPLACEMENT {Numéro de dressage::<Property>animal concerné}
{<Class>instrument} REMPLACEMENT {Numéro musical::<Property>instrument utilisé}

Différences univoques combinées des relations :
{<Generalization>(membres du personnel<--chauffeurs de poids lourds)} VOID {<Generalization>(Membre du personnel du
cirque<--Chauffeur de poids lourd)}
{<Generalization>(membres du personnel<--artistes)} VOID {<Generalization>(Membre du personnel du cirque<--Artiste)}
{<Association>(numéros---artistes)} VOID {<Association>assure(Artiste---Numéro)}
{<Association>peut nécessiter(numéros---accessoires)} VOID {<Association>nécessite(Numéro---Accessoire)}
{<Association>sont rangés(accessoires---camions)} VOID {<Association>est rangé dans(Camion---Accessoire)}
{<Association>est assigné(chauffeurs---camions)} VOID {<Association>est assigné(Camion---Chauffeur de poids lourd)}
{<Generalization>(numéros<--musicaux)} VOID {<Generalization>(Numéro<--Numéro musical)}
{<Generalization>(numéros<--dressage)} VOID {<Generalization>(Numéro<--Numéro de dressage)}

Différences univoques combinées des propriétés et opérations :
{camions::<Property>capacité} VOID {Camion::<Property>capacité}
{camions::<Property>numéro d immatriculation} VOID {Camion::<Property>numéro d immatriculation}
{camions::<Property>modèle} VOID {Camion::<Property>modèle}
{camions::<Property>marque} VOID {Camion::<Property>marque}
{accessoires::<Property>désignation} VOID {Accessoire::<Property>désignation}
{accessoires::<Property>numéro de série} VOID {Accessoire::<Property>numéro de série}
{accessoires::<Property>volume} VOID {Accessoire::<Property>volume}
{accessoires::<Property>couleur} VOID {Accessoire::<Property>couleur}
{numéros::<Property>nom} VOID {Numéro::<Property>nom}
{numéros::<Property>durée} VOID {Numéro::<Property>durée}
{numéros::<Property>code} VOID {Numéro::<Property>code}
{numéros::<Property>nombre d artistes} VOID {Numéro::<Property>nombre d artistes présents sur scène}
{artistes::<Property>pseudonymes des artistes} VOID {Artiste::<Property>pseudonymes}
{membres du personnel::<Property>numéro} VOID {Membre du personnel du cirque::<Property>numéro INSEE}
{membres du personnel::<Property>date de naissance} VOID {Membre du personnel du cirque::<Property>date de
naissance}
{membres du personnel::<Property>nom} VOID {Membre du personnel du cirque::<Property>nom}
{membres du personnel::<Property>prénom} VOID {Membre du personnel du cirque::<Property>prénom}
{membres du personnel::<Property>salaire} VOID {Membre du personnel du cirque::<Property>salaire}
{chauffeurs de poids lourds::<Property>numéro du permis de conduire} VOID {Chauffeur de poids
lourd::<Property>numero du permis de conduire}
{<Class>animal} REMPLACEMENT {Numéro de dressage::<Property>animal concerné}
{<Class>instrument} REMPLACEMENT {Numéro musical::<Property>instrument utilisé}
```

**Différences univoques unitaires des motifs simples :**
  OMISSION {<Association>utilisé(Artiste---Accessoire)}
  OMISSION {<Generalization>(Numéro<--Numéro d acrobatie)}
  OMISSION {<Class>Numéro d acrobatie}
  OMISSION {Numéro d acrobatie::<Property>type}
{<Association>utilisé pour(musicaux---instrument)} INSERTION
{<Association>concerné par(dressage---animal)} INSERTION

**Différences univoques unitaires des motifs simples masquées par les différences univoques combinées :**
  OMISSION {Numéro musical::<Property>instrument utilisé}
  OMISSION {Numéro de dressage::<Property>animal concerné}
{<Class>animal} INSERTION
{<Class>instrument} INSERTION

**Différences univoques spécifiques à l'abstraction des motifs simples :**

**Différences univoques spécifiques à la nature des motifs simples :**
{<Class>animal} NATURE_INCOMPATIBLE  PRIVE_EN_PUBLIC {Numéro de dressage::<Property>animal concerné}
{<Class>instrument} NATURE_INCOMPATIBLE  PRIVE_EN_PUBLIC {Numéro musical::<Property>instrument utilisé}

**Différences univoques spécifiques au sens des relations :**

**Différences univoques spécifiques au type d'agrégation des associations :**

**Différences univoques spécifiques des attributs et des fins d'associations :**
{camions_est assigné::<Property>} NON_NOMME_EN_NOMME MULTIPLICITES_COMMENCENT {Camion_est assigné::<Property>équipe}
{chauffeurs_est assigné::<Property>} MULTIPLICITES_TERMINENT {Chauffeur de poids lourd_est assigné::<Property>}
{camions_sont rangés::<Property>} MULTIPLICITES_COMMENCENT {Camion_est rangé dans::<Property>}
{numéros_peut nécessiter::<Property>} MULTIPLICITES_PENDANT {Numéro_nécessite::<Property>}

---------- **Différences pédagogiques ----------**
**Différences pédagogiques combinées :**
OC {<Class>Numéro d acrobatie}
 ---> PARAMS types: [] [classe], noms: [] [Numéro d acrobatie]
| OR {<Generalization>(Numéro<--Numéro d acrobatie)}
|  ---> PARAMS types: [] [héritage], noms: [] []
| OA {Numéro d acrobatie::<Property>type}
|  ---> PARAMS types: [] [attribut], noms: [] [type]

MRC incompatible {<Class>animal} {Numéro de dressage::<Property>animal concerné}
 ---> PARAMS types: [classe] [attribut], noms: [animal] [animal concerné]
| AR {<Association>concerné par(dressage---animal)}
|  ---> PARAMS types: [association binaire] [], noms: [concerné par] []

MRC incompatible {<Class>instrument} {Numéro musical::<Property>instrument utilisé}
 ---> PARAMS types: [classe] [attribut], noms: [instrument] [instrument utilisé]
| AR {<Association>utilisé pour(musicaux---instrument)}
|  ---> PARAMS types: [association binaire] [], noms: [utilisé pour] []

**Différences pédagogiques non combinées :**
OR {<Association>utilisé(Artiste---Accessoire)}
 ---> PARAMS types: [] [association binaire], noms: [] [utilisé]

DC (2 void 0 transferts) {<Class>chauffeurs de poids lourds <Class>chauffeurs} {<Class>Chauffeur de poids lourd}
 ---> PARAMS types: [classe] [classe], noms: [chauffeurs de poids lourds, chauffeurs] [Chauffeur de poids lourd]

MRFA MULTIPLICITES_COMMENCENT {camions_est assigné::<Property>} {Camion_est assigné::<Property>équipe}
 ---> PARAMS types: [fin d'association] [fin d'association], noms: [1] [1..3]

MRFA MULTIPLICITES_TERMINENT {chauffeurs_est assigné::<Property>} {Chauffeur de poids lourd_est assigné::<Property>}
 ---> PARAMS types: [fin d'association] [fin d'association], noms: [1] [0..1]

MRFA MULTIPLICITES_COMMENCENT {camions_sont rangés::<Property>} {Camion_est rangé dans::<Property>}
 ---> PARAMS types: [fin d'association] [fin d'association], noms: [1] [1..*]

MRFA MULTIPLICITES_PENDANT {numéros_peut nécessiter::<Property>} {Numéro_nécessite::<Property>}
 ---> PARAMS types: [fin d'association] [fin d'association], noms: [1] [0..*]

# Annexe 9 : Fichier de configuration du module ACDC

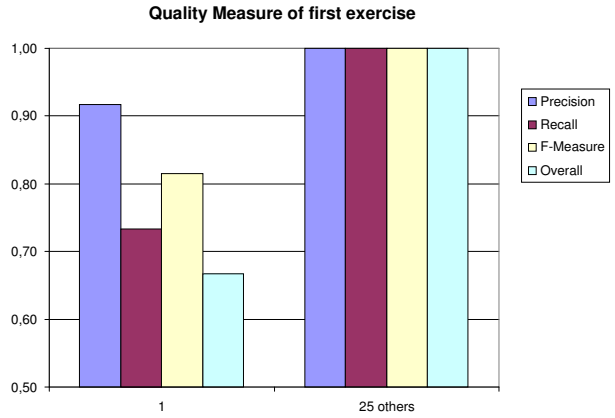
```
#############################################################################################
#                                                                                           #
# Fichier de configuration du module de ACDC de Diagram                                     #
# ACDC : Automatic Class Diagrams Comparator                                                #
#                                                                                           #
# @version v1.32                                                                            #
# @date 01/09/2008                                                                          #
# @author Ludovic Auxepaules                                                                #
#                                                                                           #
#############################################################################################

#
# 0. Options de sauvegarde d'un diagramme jGraph dans le formalisme du plugin uml2
#

# répertoire de sauvegarde des diagrammes construits par l'apprenant
sDefaultSaveDirectory=trace//
# répertoire de stockage des diagrammes de références de l'enseignant
sDefaultRefDiagramsDirectory=data//references
# extension des diagramme de classes au format uml2
sUml2DiagramFileExtension=.uml
#extension des fichiers contenant les expressions sélectionnées dans l'énoncé
sExpressionsFileExtension=.properties
# multiplicité minimale et maximale par défaut des propriétés (attributs classiques ou fin
d'association)
associationLowerEndDefault=1
associationUpperEndDefault=1
# type par défaut des propriétés (attributs classiques ou fin d'association)
sDefaultPrimitiveTypeName=anonymous
# visibilité par défaut des propriétés (attributs classiques ou fin d'association)
# une valeur au choix parmi :
sDefaultVisibilityKind=

#
# 1. Options générale du module ACDC
#

# Chemin pour accéder au jar du plugin uml2
uml2ResourcesPath=lib//uml2.uml.resources.jar
uml2PluginPath=lib//uml2.uml.jar
# nom du paquetage par défaut d'un diagramme de classes
defaultPackageName=default
# nom par défaut d'un diagramme de classes
sDefaultClassDiagramName=ClassDiagram
# nombre de niveaux de différences présentables
# de 0 à n (où n est un entier supérieur ou égal à zéro)
nbNivDifferences=6
# langue utilisée pour afficher les différences 1 : français , 0 : anglais
language=1
# booléen pour lancer en parallèle des diagnostics
bMultiDiagnotics=false
# extraction filaire des motifs complexes
# true ou false
bExtractionFilaire=false
# extraction globale des motifs complexes
# true ou false
bExtractionGlobale=true
# booléen permettant de savoir si l'on doit recomparer les motifs dont la comparaison est incomplète
à chaque nouvelle comparaison de motifs complexes
bMSRecompaisonInNewMCComparaison=false
# booléen permettant de prendre en compte de manière précise le score de chaque paire d'éléments lors
d'un appariement multivoque.
bLimitMultivoqueDifferences=true
#booléen permettant d'afficher le type des éléments dans le nom des motifs simples appariés.
bWithCompleteTypeElement=true
#booléen permettant d'afficher le nom des éléments conteneur dans le nom des motifs simples appariés.
bWithConteneurName=true
# booléen permettant de savoir si l'on prend en compte le score des motifs complexes dans le score
d'appariement des diagrammes de classes
bWithScoreMC=false
# booléen pour afficher les détails des appariements des motifs complexes
bWithComplexMotifsResults=false
#booléen permettant d'afficher à l'écran une trace des différences spécifiques
bAfficheDifferencesSpecifiques=false
#booléen permettant d'afficher à l'écran une trace des différences des élements liés ou contenus dans
les différences
bAfficheDifferencesLiees=false
#booléen permettant d'afficher à l'écran une trace des résultats du diagnostic
bAfficheResultats=true
# booléen pour afficher les appariements classés par type(si bAfficheResultats est à false, il n'y
aura pas d'affichage des statistiques également).
bAfficheAppariementsClasses=false
# booléen pour afficher les statistiques du module diagnostic (si bAfficheResultats est à false, il
n'y aura pas d'affichage des statistiques également).
bWithStatistics=false
```

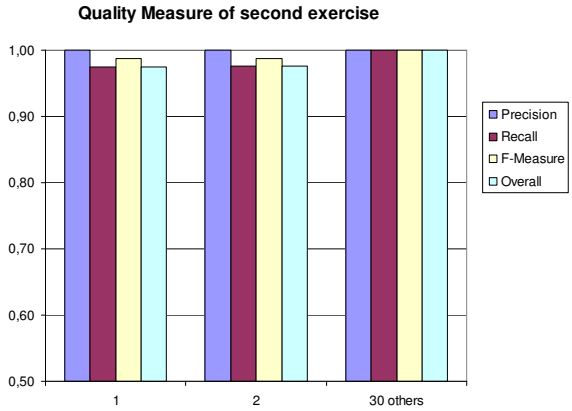
```
# booléen pour afficher le temps de calcul
bAfficheComputationalTime=true
# booléen pour afficher sur la sortie standard les traces de débogage
bDebug=false

#
# 2. Options de propagation et de dérivation
#

# Option de propagation des propriétés, des opérations, des relations des classificateurs par
l'héritage des classificateurs pères vers les fils ou/et des fils vers les pères
# @see Enum diagram.diagnostic.metaknowledge.GeneralizationPropagations
# une option au choix parmi NO_PROPAGATION , ALL_PROPAGATION , PROPAGATION_BOTTOM_UP ,
PROPAGATION_TOP_DOWN, PROPAGATION_FRATER
generalizationPropagationOptions=ALL_PROPAGATION
# Options de dérivation des associations
associationDerivationOptions=

#
# 3. Options de relaxation
#

# Les options de relaxations des associations
# une option au choix parmi la liste suivante :
# RELAXATION_COMPOSITION_TO_AGGREGATION , RELAXATION_AGGREGATION_TO_ASSOCIATION ,
RELAXATION_COMPOSITION_TO_AGGREGATION_OR_ASSOCIATION , RELAXATION_ASSOCIATION_TO_AGGREGATION ,
RELAXATION_AGGREGATION_TO_COMPOSITION , RELAXATION_ASSOCIATION_TO_AGGREGATION_OR_COMPOSITION ,
ALL_RELAXATIONS , NO_RELAXATION
associationRelaxationOptions=ALL_RELAXATIONS
# Les options de relaxations des visibilités des éléments nommés
# une option au choix parmi la liste suivante :
# , ALL_RELAXATIONS , NO_RELAXATION
visibilityRelaxationOptions=ALL_RELAXATIONS
# Les options de relaxations des multiplicités des propriétés
# une option au choix parmi la liste suivante :
# , ALL_RELAXATIONS , NO_RELAXATION
multiplicityRelaxationOptions=ALL_RELAXATIONS

#
# 4. Options de validateur syntaxique
#

syntaxicValidatorOptions=ALL

#
# 5. Options de comparaison et d'appariement de chaines de caractères
#

stringComparatorTransformations=IGNORE_CASE,IGNORE_NOMBRE,IGNORE_GENRE,IGNORE_ACCENTS,IGNORE_SPECIAL_
CHARACTERS,IGNORE_SEPARATORS
separators= \',.,;?,!,;,-,_,°
whiteSpaces= \t,\n, ,\f,\r
specialCharacters= ",*,/,\\,+,=,>,<,%,{,[,(,},],),µ,£,§,^,#,|,&,~,`,@,²,$,¤

#
# 6. Options de comparaison et d'appariement
#

# Nombre de motifs que l'on conserve pour chaque motif lors de leur comparaison
nombreDeMotifsLesPlusProches=10
# Ordre de comparaison et d'appariement :
# une option au choix parmi : BOTTOM_UP, TOP_DOWN, COMBINED
#
orderComparaisonAndMatchingOptions= TOP_DOWN
# booléen permettant de savoir s'il on ne compare que les attributs (true) ou toutes les propriétés
des classificateurs (false)
classifierbOnlyAttributesComparaison=true


#
# 7. Options des différences structurelles
#

#
# Options au choix parmi la liste suivante :
# INSERTION , OMISSION , TRANSFER , REPLACEMENT , VOID
#
matchableTransformations=INSERTION,OMISSION,TRANSFER,REPLACEMENT,VOID
#
# Options au choix parmi la liste suivante :
# UPPER , LOWER, BROTHER
#
arbreDHeritageTransformations=UPPER,LOWER,BROTHER
#
# Options au choix parmi la liste suivante :
# COMPOSITION_TO_AGGREGATION , AGGREGATION_TO_ASSOCIATION , COMPOSITION_TO_ASSOCIATION ,
ASSOCIATION_TO_AGGREGATION , AGGREGATION_TO_COMPOSITION , ASSOCIATION_TO_COMPOSITION
#
```

```
chaineDAssociationTransformations=COMPOSITION_TO_AGGREGATION,AGGREGATION_TO_ASSOCIATION,COMPOSITION_T
O_ASSOCIATION,ASSOCIATION_TO_AGGREGATION,AGGREGATION_TO_COMPOSITION,ASSOCIATION_TO_COMPOSITION

#
# 8. Options des différences pédagogiques : une différence pédagogique est constituée d'un préfixe,
#    d'un corps et d'un suffixe optionnel

# Options au choix parmi la liste suivante :
# O,A,DT,FT,T,D,F,MT,MR,MO,I,MF
#
errorPrefixes=O,A,DT,FT,T,D,F,MF,MT,MR,MO,I
#
# Options au choix parmi la liste suivante :
# C,R,A,O,FA
#
errorRoots=C,R,A,O,FA
#
# Options au choix parmi la liste suivante :
# CCA,CAC,ACAACA,ACAH,HACA,AC,AS,SA,SC,CA,CS,AH,SH,CH,HA,HS,HC,P,C,S
#
errorSuffixes=CCA,CAC,ACAACA,ACAH,HACA,AC,AS,SA,SC,CA,CS,AH,SH,CH,HA,HS,HC,P,C,S

#
# 9. Poids de chacun des critères entrant en compte dans les comparaisons des motifs
#

# pour les diagrammes de classes
classDiagramNbRootsComplexMotifs=1
classDiagramNbChildrenSimpleMotifs=1
# pour tous les les motifs
coefficientNormalisation=3
coefficientMultiplicateur=13
rapportConcordanceMax=1000000000
rapportConcordanceParfait=0
nbSubDifferences=1
matchableOmission=0
matchableInsertion=5
matchableTransfert=0
matchableRemplacement=0
matchableDuplication=1
matchableInversion=1
matchableCoefDiviseur=13
# pour les motifs simples seulement
maxNatureDistance=260
elementNature=1
elementOrphaned=0
elementNamePresence=0
elementNbRelations=31
elementNbBindComplexeMotifs=13
namedElementName=9
namedElementVisibility=9
stringComparatorCommonLettersPercent=0.8
classifierIsAbstract=9
classifierIsExtremite=26
classifierNbAttributs=7
classifierNbOperations=7
propertyAggregationKind=26
propertyType=0
propertyIsDerived=13
propertyConteneur=52
propertyMultiplicity=13
associationIsDerived=13
associationNbMemberEnds=13
relationshipMemberEnds=4
relationshipNbExtremitesPositionCoef=26
# pour les motifs complexes seulement
complexMotifNbChildren=1
complexMotifNbParents=1
complexMotifNbLinkedComplexMotifs=1
complexMotifNbClassifiers=8
complexMotifNbLinkedClassifiers=3
complexMotifNbRelationships=8
complexMotifNbInternalRelationships=3
complexMotifNbExternalRelationships=3
arbreDHeritageNbRoots=3
arbreDHeritageNbLeafs=3
arbreDHeritageNbClassesDHeritageMultiple=1
arbreDHeritageNbClassesDHeritageMultipleLosange=1
chaineDAssociationNbExtremites=3
chaineDAssociationNbMultiplesDAssociation=1
chaineDAssociationNbClassesAssociationsMultiples=1
chaineDAssociationNbAssociationsReflexives=1
chaineDAssociationNbClassesDAssociation=1
chaineDAssociationNbClassesNaires=1
```

# Annexe 10 : Résultats détaillés de l'évaluation hors-ligne d'ACDC

Dans cette annexe, nous présentons les résultats de l'évaluation hors-ligne de l'ACDC sous forme d'histogrammes pour chacun des trois exercices. Les diagrammes sont numérotés au niveau des axes des abscisses.




Pour les deux premiers exercices ayant servi à étalonner ACDC, les résultats sont excellents. Les résultats pour le dernier exercice sans changement du paramétrage d'ACDC sont bons en rappel et très bons en précision :

- Pour le premier exercice, en faisant une moyenne des 26 diagrammes étudiés, la moyenne de précision est 0,99, de rappel est 0,98, de F-Mesure est 0,99 et enfin d'overall est 0,98. Seul un diagramme n'a pas une valeur de 1 pour les quatre mesures : 0,91 en précision, 0,73 en rappel, 0,81 en F-Mesure et 0,66 en overall. Par diagramme, 141 comparaisons de motifs sont réalisées en moyenne par l'ACDC. Le temps moyen de calcul est de 0,3s.
- Pour le second exercice, en faisant une moyenne des 32 diagrammes étudiés, la moyenne de précision est 1, de rappel est 0,98, de F-Mesure est 0,99 et enfin d'overall est 0,99. Seuls deux diagramme n'ont pas une valeur de 1 pour les quatre mesures. Les valeurs minimum sont de 1 en précision, 0,97 en rappel, 0,98 en F-Mesure et 0,97 en overall. Par diagramme, 768 comparaisons de motifs sont réalisées en moyenne par l'ACDC. Le temps moyen de calcul est de 1,5s.
- Pour le troisième exercice, en faisant une moyenne des 26 diagrammes étudiés, la moyenne de précision est 0,95, de rappel est 0,85, de F-Mesure est 0,90 et enfin d'overall est 0,80. Les valeurs minimum sont de 0,81 en précision, 0,7 en rappel, 0,79 en F-Mesure et 0,59 en overall. Par diagramme, 614 comparaisons de motifs sont réalisées en moyenne par l'ACDC. Le temps moyen de calcul est de 2s.

# Table des illustrations

# TABLE DES ILLUSTRATIONS

# Glossaire
# des acronymes et des abréviations

# GLOSSAIRE DES ACRONYMES ET DES ABBRÉVIATIONS

| | |
|---|---|
| **ACDC** | **A**utomatic **C**lass **D**iagrams **C**omparator. |
| **ACM** | **A**utomated **C**ognitive **M**odeling. |
| **ACO** | **A**nt **C**olony **O**ptimization. |
| **AIED** | **A**rtificial **I**ntelligence in **ED**ucation. |
| **API** | **A**pplication **P**rogramming **I**nterface. |
| **BDOO** | **B**ases de **D**onnées **O**rientées **O**bjet. |
| **CAPIT** | **C**apitalisation **A**nd **P**unctuation **I**ntelligent **T**utor. |
| **CBM** | **C**onstraint-**B**ased **M**odelling. |
| **CIN** | **C**urriculum **I**nformation **N**etwork. |
| **CIMEL** | **C**ollaborative, constructive, **I**nquired-based **M**ultimedia **E**-**L**earning. |
| **COMA** | **CO**mbination of **M**atching **A**lgorithms. |
| **COO** | **C**onception **O**rientée **O**bjet. |
| **CROSI** | **C**apturing **R**epresenting and **O**perationalising **S**emantic **I**ntegration. |
| **CSCL** | **C**omputer-**S**upported **C**ollaborative **L**earning. |
| **DAG** | **D**irected **A**cyclic **G**raph. |
| **DBN** | **D**ynamic **B**ayesian **N**etwork. |
| **DC** | **D**iagramme de **C**lasses. |
| **DEUST ISR** | **D**iplôme d'**É**tudes **U**niversitaires **S**cientifiques et **T**echniques en **I**nformatique, **S**ystèmes et **R**éseaux. |
| **DP** | **D**ifférences **P**édagogiques. |
| **DS** | **D**ifférences **S**tructurelles. |
| **DTD** | **D**ocument **T**ype **D**efinition. |
| **EAO** | **E**nseignement **A**ssisté par **O**rdinateur. |
| **EIAH** | **E**nvironnements **I**nformatiques pour l'**A**pprentissage **H**umain. |
| **EIAO** | **E**nvironnements **I**nteractifs d'**A**pprentissage avec **O**rdinateur. |
| **EMF** | **E**clipse **M**odeling **F**ramework. |
| **EMP** | **E**clipse **M**odeling **P**roject. |
| **KERMIT** | **K**nowledge-based **E**ntity **R**elationship **M**odelling **I**ntelligent **T**utor. |
| **IA** | **I**ntelligence **A**rtificielle. |
| **I&C** | **I**nteraction et **C**onnaissance. |
| **IDE** | **I**ntegrated **D**evelopment **E**nvironment. |
| **IDL** | **I**nteractive **D**ata **L**anguage. |
| **IDM** | **I**ngénierie **D**irigée par les **M**odèles. |
| **IHM** | **I**nterface **H**omme-**M**achine. |

**ILS** **I**ndex of **L**earning **S**tyle.

**ITS** **I**ntelligent **T**utoring **S**ystem.

**LIUM** **L**aboratoire d'**I**nformatique de l'**U**niversité du **M**aine.

**MCWPA** **M**oving **C**ontracting **W**indow **P**attern **A**lgorithm.

**MDA** **M**odel **D**riven **A**rchitecture.

**MDT** **M**odel **D**evelopment **T**ools.

**MEDD** **M**ulstrategy **E**rror **D**etection and **D**iscovery.

**MOF** **M**eta-**O**bject **F**acility.

**MOO** **M**odélisation **O**rientée **O**bjet.

**MVC** **M**odel-**V**iew-**C**ontroller.

**NORMIT** **NORM**alisation **I**ntelligent **T**utor.

**NTIC** **N**ouvelles **T**echnologies de l'**I**nformation et de la **C**ommunication.

**OCL** **O**bject **C**onstraint **L**anguage.

**OMG** **O**bject **M**anagement **G**roup.

**OMT** **O**bject-**M**odeling **T**echnique.

**OOA** **O**bject-**O**riented **A**nalysis.

**OOAD** **O**bject-**O**riented **A**nalysis and **D**esign.

**OOD** **O**bject-**O**riented **D**esign.

**OOSE** **O**bject-**O**riented **S**oftware **E**ngineering.

**OWL** **W**eb **O**ntology **L**anguage.

**POO** **P**rogrammation **O**rientée **O**bjet.

**RB** **R**éseau **B**ayésien.

**RP** **R**econnaissance de **P**lans.

**SGBD** **S**ystème de **G**estion de **B**ases de **D**onnées.

**SMA** **S**ystème **M**ulti-**A**gents.

**SQL** **S**tructured **Q**uery **L**anguage.

**SSNC** **S**um of the **S**quare of the **N**umber of the same **C**haracters.

**STI** **S**ystème **T**utoriel **I**ntelligent.

**TD** **T**ravaux **D**irigés.

**TDP** **T**axonomie des **D**ifférences **P**édagogiques.

**TDS** **T**axonomie des **D**ifférences **S**tructurelles.

**TICE** **T**echnologies de l'**I**nformation et de la **C**ommunication à l'**É**ducation.

**TP** **T**ravaux **P**ratiques.

**UML** **U**nified **M**odeling **L**anguage.

**XML** e**X**tensible **M**arkup **L**anguage.

**XMI** **X**ML **M**etadata **I**nterchange.

# Références bibliographiques

# RÉFÉRENCES BIBLIOGRAPHIQUES

[Allen 1983]
Allen J.F., *Maintaining Knowledge about Temporal Intervals*, In: Communications of the ACM, vol. 26, n°11, 832-843, novembre 1983.
http://portal.acm.org/ft_gateway.cfm?id=358434&type=pdf&coll=GUIDE&dl=GUIDE&CFID=42553220&CFTOKEN=10773240

[Alonso 2009]
Alonso M., *Conception de l'interaction dans un EIAH pour la modélisation orientée objet*, Thèse de doctorat en Informatique du LIUM (Laboratoire d'Informatique de l'Université du Maine), Université du Maine, Le Mans, France, 163 pages, septembre 2009.

[Alonso *et al.* 2008]
Alonso M., Py D., Lemeunier T., *A Learning Environment for Object-Oriented Modeling, Supporting Metacognitive Regulations*, In: Proceedings of IEEE International Conference on Advanced Learning Technologies (ICALT'08), Santander, Espagne, 69-73, juillet 2008.

[Alonso & Py 2009]
Alonso M., Py D., *Une évaluation des rétroactions dans DIAGRAM, un EIAH pour la modélisation orientée objet*, In: Actes de la 4ème conférence EIAH'09, Le Mans, France, 61-68, juin 2009.
http://www-lium.univ-lemans.fr/~alonso/Documents/ALONSO_PY_EIAH09.pdf

[Anderson & Raiser 1985]
Anderson J.R., Raiser B., *The LISP tutor*, In: Byte, vol. 10, n°4, 159-175, avril 1985.
http://www.psychology.nottingham.ac.uk/staff/com/c8clat/resources/TheLISPTutor.pdf

[ArgoUML]
Tigris.org, Open Source Software Engineering Tools, *ArgoUML*, http://argouml.tigris.org/, 2009.

[Audibert 2008]
Audibert L., *Cours en ligne : UML 2*, http://laurent-audibert.developpez.com/Cours-UML/, 2008.

[Aussenac-Gilles & Charlet 2000]
Aussenac-Gilles N., Charlet J., *Ingénierie des connaissances - Modélisation*, 2000.
http://www.irit.fr/GRACQ/IMG/pdf/IC-Modelisation00.pdf

[Baghaei & Mitrovic 2005]
Baghaei N., Mitrovic A., *COLLECT-UML: Supporting individual and collaborative learning of UML class diagrams in a constraint-based tutor*, In: Rajiv Khosla, Robert J. Howlett, Lakhmi C. Jain (Eds.), Proceedings KES, Springer-Verlag, LCNS 3684, 458-464, 2005.
http://dx.doi.org/10.1007/11554028_64

[Baghaei *et al.* 2006]
Baghaei N., Mitrovic A., Irwin W., *Problem-Solving Support in a Constraint-based Tutor for UML Class Diagrams*, In Technology Instruction, Cognition and Learning (TICL 2006), vol. 4, n°2, 113-137 ,2006.
http://nilufar.baghaei.googlepages.com/Baghaei_TICL_journal.pdf

[Baghaei 2007]
Baghaei N., *A Collaborative Constraint-Based Intelligent System for Learning Object-Oriented Analysis and Design Using UML*, *Philosophiæ Doctor* thesis in Computer Science, University of Canterbury, New Zealand, 234 pages, 2007.
http://www.cosc.canterbury.ac.nz/research/reports/PhdTheses/2007/phd_0704.pdf

[Bezivin 2005]
Bézivin J., *On the Unification Power of Models*, In: Software and Systems Modeling, vol. 4, n°2, 171-188, mai 2005.
http://www.sciences.univ-nantes.fr/lina/atl/www/papers/OnTheUnificationPowerOfModels.pdf

[Bisson 1992]
Bisson F., *Learning in FOL with similarity measure*, In: Proceedings of the 10th American Association for Artificial Intelligence conference (AAAI 92), San Jose, Morgan Kaufmann, 82-87,1992.
http://www-leibniz.imag.fr/Apprentissage/Publications/Bisson-AAAI-92.pdf

[Booch 1991]
Booch, G., *Object-Oriented Design with Applications*, Benjamin/Cummings Publishing Compagny Inc. (Eds.),Redwood City, California, 580 pages, 1991.

[Booch *et al.* 1998]
Booch G., Jacobson C., Rumbaugh J., *The Unified Modeling Language - a reference manual,* Addison Wesley (Eds.), 550 pages, 1998.

[Boudreault *et al.* 2007]
Boudreault Y., Stoykova A., *À la recherche de caractéristiques d'un modèle de l'apprenant des concepts fondamentaux de la programmation (CFP)*, In : Actes de la 3ème conférence en EIAH, Lausanne, Suisse, 479-484, juin 2007.
http://hal.archives-ouvertes.fr/docs/00/16/14/89/PDF/53.pdf

[Bouhineau *et al.* 2003]
Bouhineau D., Bronner A., Chaachoua H., Huguet T., *Analyse didactique de protocoles obtenus dans un EIAH en algèbre*, In: Actes de la conférence EIAH 2003, Strasbourg, France, 79-90, avril 2003.
http://hal.archives-ouvertes.fr/docs/00/19/00/83/PDF/Bouhineau_2003.pdf

[Brown & Burton 1978]
Brown J. S., Burton R. R., *Diagnostic Models for procedural bugs in basic mathematical skills*, In: Cognitive Science vol. 2, 155-191, 1978.

[Brown 1987]
Brown A. L., *Metacognition, executive control, self-regulation and other more mysterious mechanisms*, In: F.E Weinert and R.H. Kluwe (Eds.), Metacognition, Motivation and Understanding, chapter 3: 65-116, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1987.

[Bruillard 1997]
Bruillard E., *Les machines à enseigner*, Editions Hermès, Paris, 320 pages, 1997.

[Burton 1982]
Burton R. R., *Diagnosing bugs in a simple procedural skill*, In: Intelligent Tutoring Systems, D. Sleeman D. and Brown J.S. (Eds.), Academic Press, London, 157-184, 1982.

[Burton & Brown 1982]
Burton R. R., Brown J. S., *An investigation of computer coaching for informal learning activities,* In: Intelligent Tutoring Systems, Sleeman D. and Brown J.S. (Eds.), Academic Press, New York , 79-98, 1982.

[Champin & Solnon 2003]
Champin P.-A., Solnon C., *Measuring the similarity of labeled graphs*, In: the 5th International Conference on Case-Based Reasoning (ICCBR 2003), Lecture Notes in Artificial Intelligence 2689, Springer-Verlag, 80-95, 2003.
http://www710.univ-lyon1.fr/~csolnon/publications/ICCBR03.pdf

[Cimel]
*Cimel Project*, http://www.cse.lehigh.edu/~cimel/, 2005.

[Clancey 1983]
Clancey W. J., *GUIDON*, In: Journal of Computer-Based Instruction, vol. 10, n°1, 8-14, 1983.

[Conati *et al.* 2002]
Conati C., Gertner A.,VanLhen K., *Using Bayesian Networks to Manage Uncertainty in Student Modeling*, In: User Modeling and User-Adapted Interaction, vol.12, 371-417, Kluwer Academic Publishers, NederLands, 2002.
http://www.pitt.edu/~vanlehn/distrib/umuai2002.pdf

[Crampes 2003]
Crampes J.B., *Génie logiciel - Méthode orientée-objet intégrale MACAO, Démarche participative pour l'analyse, la conception et la réalisation de logiciels*, Ellipses (Eds.), 314 pages, 2003.

[Design First]
Blank G. D., *Design First With Java*, http://designfirst.cse.lehigh.edu/, 2008.

[Dimitracopoulou *et al.*1999]
Dimitracopoulou A., Komis V., *Permettre aux élèves des activités multiples de modélisation et des approches interdisciplinaires à l'aide d'un nouveau environnement informatique*, In: Actes des XXI journées internationales sur la communication, l'éducation et la culture scientifiques et techniques, Chamonix, 243-248, 1999.

[Djoufak Kengue *et al.* 2008]
Djoufak Kengue J. F., Euzenat J., Valtchev P., *Alignement d'ontologies dirigé par la structure*, In: Actes de la 2ème Conférence Francophone sur les Architectures Logicielles (CAL 2008), Montréal, Québec, Canada., Revue des Nouvelles Technologies de l'Information RNTI-L-2, Cépaduès (Eds.), 155, mars 2008.
ftp://ftp.inrialpes.fr/pub/exmo/publications/djoufak2008a.pdf

[Do *et al.* 2002]
Do H.-H., Melnik S., Rahm E., *Comparison of Schema Matching evaluations*, In: Web, Web-Services, and Database Systems, NODe 2002 Web and Database-Related Workshops, Erfurt, Germany, October 7-10 2002, Revised Papers, LNCS 2593, Springer Berlin / Heidelberg, 221-237, 2003.
http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=5D69F067FBB9A243167864C9AB31138A?doi=10.1.1.11.4792&rep=rep1&type=pdf

[Do & Rahm 2002]
Do H.-H., Rahm E., *COMA – a system for flexible combination of schema matching approaches*, In: Proceedings of 28th International Conference on Very Large Data Bases (VLDB), Hong Kong, China, 610-621, aout 2002.
http://www.cse.ust.hk/vldb2002/VLDB2002-proceedings/papers/S17P03.pdf

[Do & Rahm 2007]
Do H.-H., Rahm E., *Matching Large Schemas: Approaches and Evaluation*, In: the Journal on Information Systems, vol. 32, n°6, 857-885, septembre 2007.
http://www.dit.unitn.it/~p2p/RelatedWork/Matching/LargeSchemas-COMA.pdf

[Doan *et al.* 2001]
Doan A., Madhavan J., Domingos P., Halevey A., *Reconciling schemas of disparate data sources: A machine Learning Approach*, In: Proceedings of ACM SIGMOD conference, Walid G. Aref (Eds.) Santa Barbara, USA, 509-520, 2001.
http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=AFB3494143C7251431F94BD6AAA71E58?doi=10.1.1.21.1104&rep=rep1&type=url&i=0

[Doise & Mugny 1984]
Doise W., Mugny G., *The social development of the intellect*, In: International Series in 881 Experimental Social Psychology, vol. 10, London, Pergamon Press, 176 pages, 1984.

[Dranidis 2007]
Dranidis D., *Evaluation of StudentUML: an Educational Tool for Consistent Modelling with UML*, In: Proceedings of the Informatics Education Europe II (IEEII), Thessaloniki, Grèce, 248-256, novembre 2007.
http://www.seerc.org/ieeii2007/PDFs/p248-256.pdf

[Eclipse]
Eclipse Foundation, *Eclipse*, http://www.eclipse.org/, 2009.

[Ehrig & Sure 2004]
Ehrig M., Sure Y., *Ontology Mapping - An Integrated Approach*, In: Proceeding of the 1st European Semantic Web Symposium, Lectures Notes in Computer Science 3053, Springer Berlin (Eds.), Heraklion, Grèce, 76-91, 2004.
http://www.aifb.uni-karlsruhe.de/WBS/meh/publications/ehrig04ontology_ESWS04.pdf

[EMF]
Eclipse Foundation, *Eclipse Modeling Framework Project (EMF)*, http://www.eclipse.org/modeling/emf/, 2009.

[EMP]
Eclipse Foundation, *Eclipse Modeling Project (EMP)*, http://www.eclipse.org/modeling/, 2009.

[Euzenat & Shvaiko 2007]
Euzenat J., Shvaiko P., *Ontology Matching*, Springer-Verlag, Berlin Heidelberg, 334 pages, 2007.

[Evans 1998]
Evans, A. S., *Reasoning with UML Class Diagrams*, In: the 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques, Florida, IEEE CS Press, 102-113, 1998.
http://www.cs.york.ac.uk/puml/papers/evanswift.pdf

[Favre *et al.* 2006]
Favre J.M., Estublier J., Blay-Fronarino M., *L'ingénierie dirigée par les modèles - Au-delà du MDA*, In: Traité IC2 - Information – Commande - Communication, Hermès Lavoisier (Eds.), 240 pages, décembre 2006.

[Felbaum 1998]
Fellbaum C., *WordNet: An Electronic Lexical Database*, Cambridge, MA, MIT Press, 423 pages, 1998.

[Felder & Silverman 1988]
Felder R. M., Silverman L. K., *Learning and Teaching Style*s In Engineering Education, In: Engineering Education 78(7), 674-681, avril 1988.
http://www4.ncsu.edu/unity/lockers/users/f/felder/public/Papers/LS-1988.pdf

[Felder 1996]
Felder R. M., *Matters of Style*, In: American Society of Engineering Education (ASEE) Prism, 6(4), 18-23, 1996.
http://www4.ncsu.edu/unity/lockers/users/f/felder/public/Papers/LS-Prism.htm

[Flavell 1977]
Flavell J. H., *Cognitive Development*., Englewood Cliffs, N.J., Prentice Hall Inc, 286 pages, 1977.

[Fowler 2003]
Fowler M., *UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd Edition*, Addison Wesley Professional (Eds.), Addison-Wesley Object (Series), 208 pages, 2003.

[Gertner & VanLehn 2000]
Gertner A., VanLhen K., *Andes: A Coached Problem Solving Environment for Physics*, In: Proceedings of the 5th International Conference on Intelligent Tutoring Systems, ITS 2000, Lectures Notes in Computer Science 1839, Springer, Berlin, 131-142, 2000.

[Gheyi *et al.* 2005]
Gheyi R., Massoni T., Borba P., *Formal Refactorings for Object Models*, In: OOPSLA'05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (Student Research Competition), San Diego, United States, 208-209, octobre 2005.
http://toritama.cin.ufpe.br/twiki/pub/SPG/GroupPublications/src-oopsla-05.pdf

[Giasson 2001]
Giasson J., *La métacognition et la compréhension en lecture*, In: Métacognition et éducation - Aspects transversaux et disciplinaires, Doudin P.-A., Martin D. et Albanese O. (Eds.), 249-262, Peter Lang, Lausanne, 2001.

[Giunchiglia *et al.* 2005]
Giunchiglia F., Shvaiko P., Yatskevich M., *Semantic schema matching*, Technical Report DIT-05-014, University of Trento, 20 pages, mars 2005.
http://eprints.biblio.unitn.it/archive/00000748/01/014.pdf

[Giunchiglia *et al.* 2007]
Giunchiglia F., Yatskevich M., Shvaiko, P., *Semantic Matching: Algorithms and Implementation*, In: The Journal on Data Semantics 9, 1-38, 2007.
http://www.cisa.informatics.ed.ac.uk/OK/Publications/Algorithms%20and%20Implementation.pdf

[Goldstein 1982]
Goldstein I. P., *The genetic graph: a representation for the evaluation of procedural knowledge*, In: Intelligent Tutoring Systems, D. Sleeman D. and Brown J. S. (Eds.), Academic Press, London, 51-78, 1982.

[Graham 1997]
Graham I, *Méthodes Orientées Objet – 2ème édition*, In : International Thomson Publishing (Eds.), France, 544 pages, 1997.

[Gruber 1993]
Gruber T. R., *A Translation Approach to Portable Ontology Specifications*, In: Knowledge Acquisition 5 (2), Academic Press Ltd., 199-220, 1993.
http://tomgruber.org/writing/ontolingua-kaj-1993.pdf

[Guéhéneuc 2003]
Guéhéneuc Y. G., *Un cadre pour la traçabilité des motifs de conception*, Thèse à l'École Nationale Supérieure des Techniques et des Mines de Nantes, Université de Nantes, 338 pages, juin 2003.
http://www.yann-gael.gueheneuc.net/Work/PhDThesis/Documents/LatestPhDThesisYannGaelGueheneuc.doc.zip

[Jacobson *et al.* 1992]
Jacobson I., Christerson M., Jonson P., Övergaard G., *Object Oriented Software Engineering: A use case driven approach*, Addison-Wesley, 552 pages, 1992.

[JGraph]
JGraph, *JGraph - The Java Open Source Graph Drawing Component*, http://www.jgraph.com/jgraph.html, 2009.

[Kalfoglou *et al.* 2005]
Kalfoglou Y., Hu B., Reynolds D., Shadbolt. N, *Capturing Representing and Operationalising Semantic Integration (CROSI) – final report*, Technical report of CROSI project, University of Southampton, 52 pages, octobre 2005.
http://eprints.ecs.soton.ac.uk/11717/1/crosi-final-report.pdf

[Kautz 1987]
Kautz H.A., *A formal theory of plan recognition*, *Philosophiæ Doctor* thesis, Department of Computer Science, University of Rochester, New Jersey, 1987.

[Kermit]
Mitrovic A., *KERMIT: a Knowledge-based Entity-Relationship Modelling Intelligent Tutoring System*, http://www.cosc.canterbury.ac.nz/tanja.mitrovic/kermit.html, 2007.

[Khayati 2005]
Khayati O., *Modèles formels et outils génériques pour la gestion et la recherche de composants*, Thèse de l'INPG (Institut National Polytechnique de Grenoble) au laboratoire LSR (Logiciels Systèmes Réseaux), Université de Grenoble, 232 pages, décembre 2005.
http://tel.archives-ouvertes.fr/docs/00/06/56/55/PDF/Oualid.khayati.these.pdf

[Kleppe *et al.* 2003]
Kleppe A., Warmer S., Bast W., *MDA explained. The Model Driven Architecture: Practice and Promise*, Addison-Wesley, 1988 pages, avril 2003.

[Komis *et al.* 2001]
Komis V., Dimitracopoulou A., Politis P., Avouris N., *Expérimentations sur l'utilisation d'un logiciel de modélisation par petits groupes d'élèves,* In: EIAO 2001, Revue STE, vol. 8, n°1-2, Hermès, Paris, 75-86, 2001.
http://modellingspace.atosorigin.es/Documents/Komis et all 2001 Experimentation ModelsCreator.pdf

[Komis *et al.* 2003]
Komis V., Avouris N., Dimitracopoulou A., Margaritis M., *Aspects de la conception d'un environnement collaboratif de modélisation à distance,* In: Actes de la conférence EIAH, Strasbourg, France, avril 2003.
http://archive-edutice.ccsd.cnrs.fr/docs/00/00/16/64/PDF/n026-141.pdf

[Kühne 2005]
Kühne T., *What is a Model?*, In: Language Engineering for Model-Driven Software Development, Dagstuhl Seminar Proceedings 040101, Bezivin J., Heckel R. (Eds.), Schloss Dagstuhl, Germany, 2005.
http://drops.dagstuhl.de/opus/volltexte/2005/23/pdf/04101.KuehneThomas1.Paper.pdf

[Kühne 2006]
Kühne T., *Matters of (Meta-) Modeling,* In: The Journal on Software and Systems Modeling, vol. 5, n°4, 369-385, décembre 2006.
http://homepages.mcs.vuw.ac.nz/~tk/publications/papers/kuehne-matters.pdf

[Kuzniarz & Staron 2005]
Kuzniarz L., Staron M., *Best Practices for Teaching UML Based Software Development*, In: Proceedings of the Educator's Symposium of MoDELS 2005, Montego Bay, Jamaica, Lecture Notes in Computer Science 3844, Springer Berlin / Heildelberg (Eds.), octobre 2005.
http://models05-edu.upb.de/proceedings/models05-edu-proceedings-a4.pdf

[Lange & Chaudron 2006]
Lange C. F. J., Chaudron M. R. V., *Effects of defects in UML Models – An experimental investigation*, In: Proceedings of the 28th International Conference on Software Engineering (ICSE'06), Shanghai China, 401-411, 2006.

[Larson *et al.* 1989]
Larson J. A., Navathe S. B., Elmasri R., *A Theory of Attributed Equivalence in Databases with Application to Schema Integration*, In: the Journal of IEEE Transactions on Software Engineering, vol. 15, n°4, 449-463, avril 1989.

[Lemeunier 2000]
Lemeunier T., *L'intentionnalité communicative dans le dialogue homme-machine en langue naturelle,* Thèse du LIUM (Laboratoire d'Informatique de l'Université du Maine), Université du Maine, 235 pages, décembre 2000.
http://www-lium.univ-lemans.fr/~lemeunie/publications/lemeunier-these2000.pdf.zip

[Lucangeli & Cornoldi 2001]
Lucangeli D., Cornoldi C., *Métacognition et mathématiques*, In: Métacognition et éducation - Aspects transversaux et disciplinaires, Doudin P.-A., Martin D. et Albanese O. (Eds.), 303-332, Peter Lang, Lausanne, 2001.

[Madhavan *et al.* 2001]
Madhavan J., Bernstein P., and Rahm E.. *Generic schema matching with Cupid*, In: Proceedings of 27th International Conference on Very Large Data Bases (VLDB), Rome, Italy, 49–58, septembre 2001.
http://db.cs.washington.edu/papers/CupidTechReport.pdf

[Magicdraw]
No Magic, *MagicDraw UML*, http://www.nomagic.com/text.php?lang=2&item=232&arg=206, 2009.

[Magnin 2006]
Magnin V., *Introduction à la modélisation,* http://magnin.plil.net/spip.php?article48, 2006.

[Malgouyres 2006]
Malgouyres H., *Définition et détection automatique des incohérences structurelles et comportementales des modèles UML - Couplage des techniques de métamodélisation et de vérification basée sur la programmation logique*, Thèse du LESIA (Laboratoire d'Etude des Systèmes Informatiques et Automatiques) à l'INSA (Institut National des Sciences Appliquées) de Toulouse, France, 151 pages, novembre 2006.
http://eprint.insa-toulouse.fr/archive/00000152/01/Malgouyres.pdf

[Massoni *et al.* 2005]
Massoni T., Gheyi R., Borba P., *Formal Refactoring for UML Class Diagrams*, In: XIX Simpósio Brasileiro de Engenharia de Software - Uberlândia, MG, Brasil, 2005.
http://twiki.cin.ufpe.br/twiki/pub/SPG/GroupPublications/sbes2005.pdf

[Mayo & Mitrovic 2001]
Mayo M., Mitrovic A., *Optimising ITS Behaviour with Bayesian Networks and Decision Theory*, In: Proceedings of the International Journal of Artificial Intelligence in Education, vol. 12, 124-153, 2001.

[MDT]
Eclipse Foundation, *Eclipse Modeling – Model Development Tools (MDT)*, http://www.eclipse.org/modeling/mdt/, 2009.

[Melnik *et al.* 2002]
Melnik S., Garcia-Molina H., Rahm E., *Similarity Flooding: Versatile Graph Matching Algorithm and its Application to Schema Matching*, In: Proceedings of the 18th International Conference on Data Engineering (ICDE 2002), San Jose, 117-128, 2002.
http://ilpubs.stanford.edu:8090/730/1/2002-1.pdf

[Mitrovic & Ohlsson 1999]
Mitrovic A., Ohlsson S., *Evaluation of a Constraint-Based Tutor for a Database Language*, In: The International Journal on Artificial Intelligence in Education, vol. 10, n°3-4, 238-56, 1999.
http://ir.canterbury.ac.nz/bitstream/10092/327/1/41178_ijaied99-paper.pdf

[Mitrovic *et al.* 2001]
Mitrovic A., Mayo M., Suraweera P., Martin B., *Constraint-based tutors: a success story*, In: Proceedings of the 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA/AIE-2001, Budapest L., Monostori J., Vancza and M. Ali (Eds.), Springer-Verlag Berlin Heidelberg, LNAI 2070, 931-940, juin 2001.
http://www.cosc.canterbury.ac.nz/tanja.mitrovic/cbmtut.pdf

[Mitrovic 2003]
Mitrovic A., *An intelligent SQL tutor on the Web*, In: Proceedings of the International Journal of Artificial Intelligence in Education, vol. 13, n°2-4, 173-197, 2003.
http://hal.archives-ouvertes.fr/docs/00/19/73/16/PDF/mitrovic03.pdf

[Mitrovic 2005]
Mitrovic A., *The Effect of Explaining on Learning: a Case Study with a Data Normalization Tutor*, In: Proceedings of the Conference on Artificial Intelligence in EDucation (AIED 2005), C.-K. Looi, G. McCalla, B. Bredeweg, J. Breuker (Eds.), IOS Press, 499-506, 2005.
http://www.cosc.canterbury.ac.nz/tanja.mitrovic/NORMIT-AIED05.pdf

[Mizoguchi & Bourdeau 2004]
Mizoguchi R., Bourdeau J., Le *rôle de l'ingénierie ontologique dans le domaine des EIAH*, In: Revue des Sciences et Technologies de l'Information et de la Communication pour l'Education et la Formation (STICEF) vol. 11, numéro spécial « Ontologies pour les EIAH », 231-246, 2004.
http://sticef.univ-lemans.fr/num/vol2004/mizoguchi-06/sticef_2004_mizoguchi_06.pdf

[MontyLingua]
Liu H., *MontyLingua V.2.1 (Python and Java), A Free, Commonsense-Enriched Natural Language Understander for English*, http://web.media.mit.edu/~hugo/montylingua/, 2004.

[Morand 1998]
Morand B., *Modeling : is it turning informal into formal,* In: International Workshop Communication UML 1998 : Beyond de notation, Mulhouse, France, http://www.iutc3.unicaen.fr/~moranb/uml/uml.html, 1998.

[Moritz & Blank 2005]
Moritz S., Blank G. D., *A Design-First Curriculum for Teaching Java in a CS1 Course*, In: The ACM SIGSE Bulletin, vol. 37, n°2, 89-93, juin 2005.
http://www.cse.lehigh.edu/cimelits/paper/sigCSEJournalPaper.pdf

[Moritz *et al.* 2005]
Moritz S., Wei F., Parvez S., Blank G. D., *From Objects-First to Design-First with Multimedia and Intelligent Tutoring*, In: Proceeding of The 10th annual SIGCSE (ACM Special Interest Group on Computer Science Education) Conference on Innovation and Technology in Computer Science Education, Caparica, Portugal, 99-103, juin 2005.
http://www.cse.lehigh.edu/cimelits/paper/ITiCSE05.pdf

[Moritz 2008]
Moritz S., *Generating and Evaluating Object-Oriented Designs in an Intelligent Tutoring System*, *Philosophiæ Doctor* thesis in Computer Science, Lehigh University, 188 pages, mars 2008.
http://designfirst.cse.lehigh.edu/MoritzDissertation.pdf

[Nicaud 1987]
Nicaud J.-F., *Aplusix : un système expert en résolution pédagogique d'exercices d'algèbre*, Thèse de Doctorat de l'Université Paris XI-Orsay, France, 189 pages, 1987.

[Nicaud & Vivet 1988]
Nicaud J.-F., Vivet M., *Les tuteurs intelligents :réalisations et tendances de recherches*, In: TSI, vol.7, n°1, numéro spécial, 21-46, 1988.

[Ohlsson 1994]
Ohlsson S., *Constraint-Based Student Modeling*, In: J. E. Greer, G. McCalla (Eds.) Student Modelling : the Key to Individualized Knowledge-based Instruction, Berlin, Springer, 167-189, 1994.

[Ohlsson 1996]
Ohlsson S., *Learning from performance errors*, In: Psychological Review, vol. 103, n°2, American Psychological Association (Eds.), Washington, DC, 241-262, 1996.

[OMG MOF]
Object Management Group, *OMG's MetaObject Facility*, http://www.omg.org/mof/, 2009.

[OMG UML]
Object Management Group, *Unified Modeling Language™, UML® Resource Page,* http://www.uml.org/, 2009.

[OMG XMI]
Object Management Group, *Catalog of OMG Modeling and Metadata Specifications - XML Metadata Interchange (XMI®)*, http://www.omg.org/technology/documents/modeling_spec_catalog.htm#XMI, 2009.

[Parvez 2007]
Parvez S. M., *A Pedagogical Framework For Integrating Individual Learning Style Into An Intelligent Tutoring System*, *Philosophiæ Doctor* thesis in Computer Science, Lehigh University, 150 pages, décembre 2007.
http://designfirst.cse.lehigh.edu/ParvezDissertation.pdf

[Parvez & Blank 2008]
Parvez S. M., Blank G. D., *Individualizing Tutoring with Learning Style Based Feedback*, In: Proceedings of the Intelligent Tutoring System (ITS 2008), Beverly Park Woolf, Esma Aïmeur, Roger Nkambou, Susanne P. Lajoie (Eds.), LNCS 5091, Springer, Montreal, Canada, juin 2008.
http://lvstem.cse.lehigh.edu/documents/PARVEZ_ITS2008.pdf

[Pearl 1984]
Pearl J., *Heuristics: Intelligent search strategies for computer problem solving*, Addison-Wesley, 382 pages, 1984

[Penders 2002]
Penders T., *Introduction à UML,* ISBN : 2-7464-0442-7, OEM (Eds.), 426 pages, 2002.

[Piechoki]
Piechocki L., *UML en français,* http://uml.free.fr/, 2006.

[Politis *et al.* 2001]
Politis P., Komis V., Dimitracopoulou A., *ModelsCreator : un logiciel de modélisation permettant l'utilisation des règles logiques et la prise de décision*, In : Revue de l'E.P.I. (Enseignement Public et Informatique), n°102, Paris, 179-199, juin 2001.
http://archive-edutice.ccsd.cnrs.fr/docs/00/04/40/64/PDF/ba2p179.pdf

[Poseidon]
Gentleware model to business, *Poseidon for UML*, http://www.gentleware.com/products.html, 2009.

[Py 1996]
Py D., *Aide à la démonstration en géométrie : le projet Mentoniezh*, In: Sciences et Techniques Educatives, vol. 3, n°2, 227-256, 1996.

[Py 2001]
Py D., *Environnements interactifs d'apprentissage et démonstration en géométrie*, Habilitation à Diriger des Recherches, Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), Rennes, 63 pages, juillet 2001.
http://hal.archives-ouvertes.fr/docs/00/19/02/02/PDF/Py_2001.pdf

[Py *et al.* 2008]
Py D., Alonso M., Auxepaules L., Lemeunier T., *Design of Pedagogical Feedbacks in a Learning Environment for Object-Oriented Modeling*, In: Promoting Software Modeling through Active Education, Proceedings of Educators Symposium of the ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS'08), Michal Smialek (Eds.), Toulouse, France, 39-50, 2008.

[Rahm & Bernstein 2001]
Rahm E., Bernstein P. A., *A survey of approaches to automatic schema matching*, In: The international Very Large DataBases Journal (VLDB Journal), vol. 10, n°4, Springer Berlin / Heidelberg, 334-350, décembre 2001.
http://www.springerlink.com/content/y3bavwk2t7328hat/fulltext.pdf

[Raimbault *et al.* 2006]
Raimbault T., Genest D., Loiseau S., *Interroger et vérifier des diagrammes de classes UML : une approche fondée sur le modèle des graphes conceptuels*, In: Actes du 15ème congrès francophone Reconnaissance des Formes et Intelligence Artificielle (RFIA'06), Tours, France, 2006.
http://www.rfai.li.univ-tours.fr/RFIA06_Events/CDRFIA/pdf/077.pdf

[Ramollari & Dranidis 2007]
Ramollari R., Dranidis D., *StudentUML: An Educational Tool Supporting Object-Oriented Analysis and Design*, In: Proceedings of the 11th Panhellenic Conference on Informatics (PCI 2007), Patras, Greece, 363-373, mai 2007.
http://pci2007.upatras.gr/proceedings/PCI2007_volA/A_363-373_Ramollari.pdf

[Rational]
IBM, *Rational Software*, http://www-306.ibm.com/software/rational/, 2009.

[Renaudie 2005]
Renaudie D., *Méthodes d'apprentissage automatique pour la modélisation de l'élève en algèbre*, Thèse de l'INPG (Institut National Polytechnique de Grenoble) au laboratoire LEIBNIZ-IMAG, Université de Grenoble, 160 pages, janvier 2005.
http://www-leibniz.imag.fr/Apprentissage/Publications/Renaudie-2005-These.pdf

[Rumbaugh *et al.* 1991]
Rumbaugh J. R., Blaha M. R., Lorensen W., Eddy F., Premerlani W., *Object–oriented modeling technique*, Prentice-Hall, New Jersey, 500 pages., 1991.

[Seuma Vidal *et al.* 2005]
Seuma Vidal J.-P., Malgouyres H., Motet G., *Règles de Cohérence UML 2.0*, Rapport interne du LESIA (Laboratoire d'Etude des Systèmes Informatiques et Automatiques), 194 pages, juillet 2005.
http://www.lesia.insa-toulouse.fr/~motet/UML/

[Seuma Vidal 2006]
Seuma Vidal J.-P., *Maîtrise de la cohérence des modèles UML d'applications critiques - Approche par l'analyse des risques liés au langage UML*, Thèse du LESIA (Laboratoire d'Etude des Systèmes Informatiques et Automatiques) à l'INSA (Institut National des Sciences Appliquées) de Toulouse, France, 131 pages, juin 2006.
http://www.lattis.univ-toulouse.fr/lesia/uploaded/files/publications/theses/theseJPSeumaVidal.pdf

[Sheth & Larson 1990]
Sheth A., Larson J., *Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases*, In: ACM Computing Surveys, vol. 22, n°3, 183-230, septembre 1990.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.6768&rep=rep1&type=pdf

[Shvaiko 2004]
Shvaiko P., *A Classification of Schema-Based Matching Approaches*, In: Proceedings of Meaning Coordination and Negotiation workshop at ISWC'04, Hiroshima, Japan, 119-130, novembre 2004.
http://eprints.biblio.unitn.it/archive/00000654/01/093.pdf

[Shvaiko & Euzenat 2005]
Shvaiko P., Euzenat J., *A Survey of Schema-based Matching Approaches*, In: The Journal on Data Semantics, vol. 4, 146-171, 2005.
http://www.dit.unitn.it/~p2p/RelatedWork/Matching/JoDS-IV-2005_SurveyMatching-SE.pdf

[Shvaiko & Euzenat 2008]
Shvaiko P., Euzenat J., *Ten Challenges for Ontology Matching*, In: Proceedings of OTM 2008 Confederated International Conferences, Part II, LNCS 5332, Robert Meersman, Zahir Tari (Eds.), Monterrey, Mexico, 1164-1182, novembre 2008.
http://www.dit.unitn.it/~p2p/RelatedWork/Matching/042.pdf

[SimMetrics]
Chapman S., Natural Language Processing Group, Department of Computer Science, University of Sheffield, *SimMetrics*, http://www.dcs.shef.ac.uk/~sam/simmetrics.html, 2006.

[Sison & Shimura 1998]
Sison R., Shimura M., *Student Modeling and Machine Learning*, In: The International Journal of Artificial Intelligence in Education, vol. 9, 128-158, 1998.
http://www.cs.ubc.ca/~conati/532b/papers/sm-ml.pdf

[Soller 2001]
Soller A., *Supporting Social Interaction in an Intelligent Collaborative Learning System*, In: The International Journal of AIED, vol. 12, 40-62, 2001.
http://aied.inf.ed.ac.uk/members01/archive/vol_12/soller/paper.pdf

[Solnon 2005]
Solnon C., *Contributions à la résolution pratique de problèmes combinatoires : des fourmis et des graphes,* Habilitation à Diriger des Recherches, Laboratoire d'InfoRmatique en Image et Systèmes d'information (LIRIS), Université Claude Bernard Lyon 1, France, 115 pages, décembre 2005.
http://www710.univ-lyon1.fr/%7Ecsolnon/publications/memoire.ps

[Sorlin 2006]
Sorlin S., *Mesurer la similarité de graphes*, Thèse du Laboratoire d'InfoRmatique en Image et Systèmes d'information (LIRIS), Université Claude Bernard Lyon 1, France, 142 pages, novembre 2006.
http://liris.cnrs.fr/sebastien.sorlin/publications/these.pdf

[Sorlin *et al.* 2006]
Sorlin S., Sammound O., Solnon C., Jolion J. M., *Mesurer la Similarité de Graphes*, In: Extraction de COnnaissance à partir d'Images (ECOI 2006), Atelier de Extraction et Gestion de Connaissances (EGC 2006), 21-30, 2006.
http://www.grappa.univ-lille3.fr/~ppreux/egc2006/actes/ecoi.pdf

[Sorlin *et al.* 2007]
Sorlin S., Solnon C., Jolion J.-M., *A Generic Graph Distance Measure Based on Multivalent Matchings*, In: Applied Graph Theory in Computer Vision and Pattern Recognition, 151–182, 2007.

[Stachoviak 1973]
Stachoviak H., *Allgemeine Modelltheorie*, Springer-Verlag, Wien and New York, 1973.

[Steinmüller 1993]
Steinmüller W., *Informationstechnologie und Gesellschaft: Einfuhrung in die Angewandte Informatik,* Wissenschaftliche, Buchgesellschaft, Darmstadt, 1993.

[Suraweera 2001]
Suraweera P., *An Intelligent Teaching System for Database Modelling*, Master thesis presented for the Master of Science (M.Sc.) degree in Computer Science, University of Canterbury , 2001.
http://www.cosc.canterbury.ac.nz/research/reports/MastTheses/2001/mast_0102.pdf

[Suraweera & Mitrovic 2002]
Suraweera P., Mitrovic A., *KERMIT : a Constraint-based Tutor for Database Modeling,* In : Proceedings of the 6th International Conference on Intelligent Tutoring Systems (ITS 2002), Biarritz, France, LCNS 2363, 377-387, 2002.
http://www.pramu.orconhosting.net.nz/papers/suraweera-its2002.pdf

[Suraweera & Mitrovic 2004]
Suraweera P., Mitrovic A., *An Intelligent Tutoring System for Entity Relationship Modeling,* In: The International Journal of Artificial Intelligence in Education, vol. 14, n°3-4, 375-417, 2004.

[Tanana *et al.* 2008]
Tanana M., Delestre N., Pecuchet J.-P., Bennouna M., *Évaluation du savoir-faire en électronique numérique à l'aide d'un algorithme de classification*, In: Actes de la conférence Technologies de l'Information et de la Communication pour l'Éducation (TICE'08), Paris, France, 44-51, octobre 2008.
http://tice2008.institut-telecom.fr/archive/45/TICE2008_Actes_web09.pdf

[Tchounikine 2002]
Tchounikine P., *Quelques éléments sur la conception et l'ingénierie des EIAH*, In: Actes du GDR $I^3$ (Information-Interaction-Intelligence), 233-245, 2002.
http://www-lium.univ-lemans.fr/~tchou/Tchou_Conception_des_EIAH_(GDR_I3).pdf

[Together]
Borland® *the Open ALM Company*, *Borland® Together®, Visual Modeling for Software Architecture Design*, http://www.borland.com/together/, 2009.

[UML2]
Eclipse Foundation, Model Development Tools (MDT) - *UML2:* an *EMF-based implementation of UML 2.x OMG metamodel for Eclipse platform*, http://www.eclipse.org/uml2/, 2009.

[UML Superstructure]
OMG, *OMG Unified Modeling Language Superstructure v2.1.2 Specification*, http://www.omg.org/docs/formal/07-11-02.pdf, 738 pages, novembre 2007.

[Van Roy]
Van Roy P., *Une taxonomie des principaux paradigmes de programmation*, Département d'ingénierie informatique de l'UCL (Université Catholique de Louvain), http://www.info.ucl.ac.be/~pvr/paradigmes.html, 2008.

[Van Roy & Haridi 2004]
Van Roy P., Haridi S., *Concepts, Techniques, and Models of Computer Programming*, MIT Press, 900 pages, 2004.

[Vivet & Baron 1987]
Vivet M., Baron M., *Systèmes experts et tuteurs intelligents*, In: 6ème congrès AFCET – Reconnaissance des formes et intelligence artificielle, Antibes, novembre 1987.

[Volle 2004]
Volle M., *À propos de la modélisation*, http://www.volle.com/travaux/modelisation2.htm, 2004.

[Wenger 1987]
Wenger E., *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*, CA: Morgan Kaufmann Publishers, Inc., Los Altos, California, 1987.

[Wey 2007]
Wey S. M., *A Student Model For an Intelligent Tutoring System Helping Novices Learn Object-Oriented Design*, *Philosophiæ Doctor* thesis in Computer Science, Lehigh University, 187 pages, août 2007.
http://designfirst.cse.lehigh.edu/FangWeiDissertation-StudentModel.pdf

[Wikipedia]
Wikipedia L'encyclopédie libre, http://fr.wikipedia.org/wiki/, 2009.

[Wiktionnaire]
Wiktionnaire Le dictionnaire libre, http://fr.wiktionary.org/wiki/, 2009.

[WordNet]
Cognitive Science Laboratory of Princeton University, *WordNet: a lexical database for the English language*, http://wordnet.princeton.edu/, 2006.

[Yang *et al.* 2002]
Yang Q. X., yuan S. S., Chun L., Rajasekera J., *An Important Issue in Data Mining-Data Cleaning*, In: Proceedings of the 16th Pacific Asia Conference, Jeju, Korea, 455-464, 2002.
http://dspace.wul.waseda.ac.jp/dspace/bitstream/2065/12252/1/PACLIC16-455-464.pdf

[Zerdazi & Lamolle 2006]
Zerdazi A., Lamolle M., *Intégration de sources hétérogènes par matching semi-automatique de schémas XML étendus*, In : Actes du 24ème congrès INFORSID, Hammamet, Tunisie, 991-1006, juin 2006.
http://134.214.81.35/articles/a626c1LqQAUUcMC4M.pdf

# Analyse des diagrammes de l'apprenant dans un EIAH de la modélisation orientée objet

## Le système ACDC


## Résumé

Nos travaux s'inscrivent dans le cadre des recherches menées sur les EIAH (Environnements Informatiques pour l'Apprentissage Humain) dans le projet Interaction et Connaissance du LIUM. Ce projet a suscité le développement de Diagram, un EIAH dédié à l'apprentissage des concepts de la modélisation orientée objet. Dans cette thèse, nous nous intéressons à l'analyse des réponses de l'apprenant lors de l'activité de modélisation de construction d'un diagramme de classes UML à partir de spécifications textuelles. En l'absence de résolveur pédagogique dans ce contexte, nous proposons une méthode d'analyse automatique des diagrammes de l'apprenant. Nous présentons un outil de diagnostic basé sur la comparaison et l'appariement des constituants de plusieurs diagrammes. Cette proposition s'inspire des concepts et des techniques d'appariement de modèles et se concentre principalement sur les aspects structurels des modèles à apparier. La méthode permet d'exprimer en sortie des appariements et des différences entre le diagramme de l'apprenant et un diagramme de référence construit par un expert. La méthode est instanciée sous forme d'un composant logiciel intégré à Diagram, nommé ACDC (*Automatic Class Diagrams Comparator*). Les résultats d'ACDC (les différences relevées) sont traités dans Diagram pour la production de rétroactions pédagogiques synchrones destinées à l'apprenant. La pertinence et la qualité des résultats produits par ACDC ont été évaluées en dehors de Diagram sur un corpus de diagrammes collectés dans des situations réelles d'apprentissage. Une expérimentation de Diagram (après l'intégration d'ACDC) a été menée fin 2008 avec des étudiants de l'Université du Maine.



**Mots clés :** EIAH, diagnostic, appariement structurel, modélisation orientée objet, diagramme de classes, UML.



## Summary

Our work is part of a project of the LIUM laboratory, which aims at designing models, methods and tools for computer-supported learning environments. Diagram, a computer-supported learning environment for UML object-oriented modeling has being developed in this project. In this thesis, we focus on the analysis of students' answers during the modeling activity of building of an UML class diagram from textual specifications. A pedagogical solver can't be use in this context. We propose an method that analyses automatically diagrams of the learner. We expose a diagnostic tool based on the comparison and matching of components of several diagrams. This proposal follows the model matching concepts and techniques and it focuses mainly on structural aspects of models to match. This method expresses differences between the student diagram and a reference diagram constructed by an expert. The proposed matching method is instantiated as a software component of Diagram environment, called ACDC (Automatic Class Diagrams Comparator). The ACDC results (the identified differences) are exploited by Diagram in order to produce synchronous pedagogical feedbacks to the learner. The relevance and quality of ACDC results has been evaluated outside Diagram on a diagrams set built by students in real situation of learning. An experimentation of Diagram (since the ACDC integration) has been conducted during Fall 2008 with students in a UML course taught at the Université du Maine.



**Keywords :** computer-supported learning environments, diagnosis, structural matching, object-oriented modeling, class diagram, UML.