CONCEPTION DE L'INTERACTION DANS UN EIAH POUR LA MODÉLISATION ORIENTÉE OBJET

THÈSE

présentée et soutenue publiquement le 23 septembre 2009

pour l'obtention du

Doctorat de l'Université du Maine
(spécialité informatique)

par

MATHILDE ALONSO

Composition du jury

Rapporteurs

M. Eric BRUILLARD	Professeur à l'IUFM de Créteil
Mme Monique GRANDBASTIEN	Professeur à l'Université Henri Poincaré, Nancy 1

Examinateurs

Mme Monique BARON	Maître de Conférences à l'Université Pierre et Marie Curie, Paris 6
M. Pascal LEROUX	Professeur à l'Université du Maine, Le Mans *président du jury*

Directrice

Mme Dominique PY	Professeur à l'Université du Maine, Le Mans

Laboratoire d'Informatique de l'Université du Maine

Remerciements

Je remercie Monique Grandbastien, Professeur à l'Université Henri Poincaré de Nancy et Eric Bruillard, Professeur à l'IUFM de Créteil, de m'avoir fait l'honneur d'être les rapporteurs de ma thèse.

Je remercie Monique Baron, Maître de conférences à l'Université Pierre et Marie Curie de Paris 6, de m'avoir fait l'honneur d'être examinatrice de ma thèse.

Je remercie Pascal Leroux, Professeur de l'Université du Maine, de m'avoir fait l'honneur d'être examinateur de ma thèse et président du jury.

Je remercie Dominique Py pour la direction de cette thèse, ainsi que Thierry Lemeunier et Ludovic Auxepaules pour le travail que nous avons mené ensemble. Je remercie également Gaël Salaün pour son travail d'ingénieur sur DIAGRAM, Pierre Laforcade qui m'a donné l'opportunité de réaliser des expérimentations à Laval, et tous les étudiants ayant participé aux différentes sessions d'expérimentations menées au cours de ces travaux de recherche.

Je tiens à remercier l'ensemble des membres du laboratoire LIUM avec qui j'ai partagé d'agréables moments et pauses café. Merci pour leur soutien moral au cours de ces quatre années. Un grand merci aux anciens et actuels doctorants du Mans pour leur présence et leur amitié : Christelle et Bérangère, mes anciennes collègues de bureau, Julie, Naïma et Johan pour leurs précieux conseils, Carole pour sa sympathie et ses relectures attentives (et littéraires !) de mon manuscrit, Aina, Vincent, Thierry, Richard et Antoine pour leur bonne humeur au quotidien. Je n'oublie pas les doctorants de Laval qu'on a trop peu l'occasion de voir ! Je pense spécialement à Hassina pour son soutien, Boubekeur et Firas pour les bons moments partagés à ICALT'08. Merci également à Bruno et Etienne qui ont toujours su m'apporter leur aide et leur expertise technique. Un merci particulier à tous ceux qui m'ont aidé le jour de la soutenance : Elodie, Richard, Naïma, JS et Vincent.

Je remercie vivement l'ensemble de mes proches et de mes amis qui, de près ou à distance, ont toujours su m'écouter et m'encourager. Merci à Séverine, my friend from Le Mans !, Delphine, Lyvia, San et Virginie pour nos week-ends que j'attends toujours impatiemment. Merci également à Angela, Romain et Annamaria qui m'ont soutenue depuis le Master EIAHD. Merci enfin à ma famille et à Vincent qui m'ont accompagnée et supportée (dans les deux sens du terme !) tout au long de ce travail.

Table des matières

CHAPITRE 1

Contexte et problématique

Sommaire

1.1 Introduction

Les travaux présentés dans ce mémoire s'inscrivent dans les recherches menées sur les Environnements Informatiques pour l'Apprentissage Humain (EIAH). Ce domaine de recherche est dédié à la conception d'environnements informatiques dans le but de favoriser l'apprentissage humain, c'est-à-dire la construction de connaissances chez un apprenant [Tchounikine, 2002].

Nous nous intéressons à l'interaction dans un environnement informatique pour l'apprentissage de la construction de modèles dans le cadre d'activités scientifiques et plus spécialement en informatique. En cela, nous nous situons dans le courant de recherche en EIAH qui considère que les interactions entre l'apprenant et le système sont au cœur du processus d'apprentissage. La connaissance à acquérir émerge alors de l'interaction entre l'apprenant et son environnement.

Nos travaux ont été réalisés dans le cadre du projet « Interaction et connaissances dans les EIAH pour la modélisation », mené au sein du Laboratoire d'Informatique de l'Université du Maine (LIUM).

1.2 Le projet « Interaction et connaissances dans les EIAH pour la modélisation »

L'objectif du projet « Interaction et connaissances dans les EIAH pour la modélisation » est de concevoir des modèles, méthodes et outils pour la conception d'environnements informatiques dédiés à l'apprentissage de la modélisation dans le domaine de l'informatique. Un modèle peut être défini comme « la représentation simplifiée d'un système ». Dans le domaine de l'informatique, un modèle fournit une représentation d'un système à l'aide d'une notation graphique constituée d'entités reliées par des relations, comme par exemple les schémas relationnels de bases de données ou les diagrammes utilisés en génie logiciel. La création de cette représentation schématique, c'est-à-dire l'activité de modélisation, constitue une part importante de l'activité scientifique des professionnels : elle requiert des compétences et des habiletés spécifiques et, de ce fait,

est apparue depuis une dizaine d'années dans les cursus d'enseignement informatique à vocation professionnelle.

Ce projet se définit selon deux axes principaux de recherche :

- La définition de modes d'interaction pertinents dans les EIAH pour la modélisation. Les objectifs sont d'identifier les activités à proposer et leurs effets respectifs sur l'apprentissage, d'élaborer des outils graphiques et langagiers pour la manipulation des représentations dans l'EIAH et de définir les rétroactions produites par l'EIAH.
- L'identification des connaissances à embarquer dans ce type d'EIAH. Il s'agit de caractériser la solution attendue (solution idéale, variantes, contraintes à respecter, etc.), d'élaborer des méthodes de validation et de diagnostic d'erreurs, et d'identifier les conceptions et comportements des apprenants en vue de construire un modèle de l'apprenant.

Les deux axes de recherche du projet sont traités au sein de deux thèses : nos travaux se situent dans le premier axe de travail, le deuxième thème faisant l'objet de la thèse de Ludovic Auxepaules. Bien que traités séparément, les deux axes de recherche du projet « Interaction & connaissances » sont liés. En effet, l'analyse et l'évaluation des productions de l'apprenant permet de produire des rétroactions adaptées à son travail.

Dans un souci de généricité, le projet « Interaction & connaissances » s'attache à élaborer des modèles, méthodes et outils indépendants du domaine de modélisation visé par l'apprentissage. Toutefois, il est nécessaire d'instancier ces éléments sur un domaine spécifique afin de tester et de valider l'approche proposée. Le projet « Interaction & connaissances » s'est naturellement porté sur la modélisation orientée objet dans le domaine du génie logiciel. En effet, cette discipline est enseignée par Thierry Lemeunier, maître de conférence en informatique à l'université du Maine, et participant au projet « Interaction & connaissances ». Thierry Lemeunier intervient au sein du projet en tant qu'expert du domaine de la modélisation orientée objet et enseignant référent concernant les choix pédagogiques pour l'enseignement de cette discipline.

Afin d'instancier le modèle sur le domaine de la modélisation orientée objet et d'articuler les modèles élaborés au sein des deux axes de recherche, le projet « Interaction & connaissances » a développé DIAGRAM, un logiciel pour l'apprentissage de la modélisation orientée objet.

1.3 Problématique et objectifs de recherche

Notre travail de recherche aborde la question de l'interaction dans un environnement informatique pour l'apprentissage de la modélisation. Notre problématique concerne plus particulièrement la conception de l'interaction dans un tel environnement. Trois questions générales découlent de cette problématique :

- quelles activités et quels types de tâches doit-on proposer à l'apprenant ?
- quelles modalités d'action doit-on offrir à l'apprenant dans un environnement d'apprentissage pour la modélisation ?
- quelles rétroactions doit fournir l'environnement ?

Concernant les activités et les types de tâches à proposer, notre objectif est d'organiser l'activité de manière à favoriser l'acquisition progressive des concepts et de varier les tâches afin de mobiliser les différents savoir-faire en jeu. La modélisation est une activité qui nécessite des savoir-faire et des connaissances de niveau métacognitif. L'activité réflexive de l'apprenant au sein d'un EIAH pour la modélisation doit donc être facilitée et supportée par des modes d'interaction et des outils d'aide spécifiques.

L'objectif de nos travaux est d'étudier et d'organiser l'interaction dans un environnement d'apprentissage de la modélisation dans le but de concevoir un modèle générique applicable à différents domaines de la modélisation informatique. Dans le projet « Interaction & connaissances » ainsi que dans nos travaux, nous étudions en particulier le cas de la modélisation orientée objet.

1.4 Méthodologie de recherche

Quatre phases se sont succédées dans notre travail : la première est une phase préliminaire et les trois suivantes correspondent aux trois grandes questions qui constituent notre problématique. Au cours de la phase préliminaire, nous avons étudié l'existant du projet « Interaction & connaissances ». Nous avons également amorcé l'étude des types d'activités qui peuvent être proposées à l'apprenant et avons mené une expérimentation exploratoire avec le prototype existant de DIAGRAM. Ce prototype intégrait des modes d'interaction spécifiques, issues de premières réflexions au sein du projet « Interaction & connaissances », que nous avons raffinés et améliorés tout au long de notre travail de recherche.

Suite à l'analyse de la tâche de modélisation et des mises à l'essai du prototype de DIAGRAM, nous avons constaté l'importance de l'aspect métacognitif dans la tâche de modélisation et cette constatation a orienté nos recherches concernant les modalités d'interaction. Nous avons donc étudié les possibilités d'intégration d'aides métacognitives dans un EIAH pour la modélisation, puis implémenté et expérimenté ces propositions.

Dans la troisième phase de notre étude, nous avons approfondi l'étude des activités que l'on peut proposer aux apprenants en réalisant notamment des expérimentations exploratoires de ces tâches en environnement informatique. Nous avons élaboré un ensemble d'activités sous forme de scénarios que nous avons implémentés dans DIAGRAM.

Enfin, la quatrième phase de notre travail a consisté en l'étude des rétroactions du système. A ce stade, nous disposions dans le projet « Interaction & connaissances » d'un outil de diagnostic, développé dans le cadre de la thèse de Ludovic Auxepaules, dont nous avons exploité les résultats. Nous avons étudié et conçu une taxonomie de rétroactions, implémenté puis expérimenté nos propositions avec DIAGRAM.

De manière transversale, notre étude a été menée selon la représentation en trois axes proposée par [Mackay et Fayard, 1997] : Théorie – Conception d'artefacts – Observation (cf. figure 1.1).

Au niveau théorique, nous avons analysé l'existant du projet, c'est-à-dire les premiers modes d'interaction proposés, et nous avons étudié la métacognition ainsi que son importance dans l'activité de modélisation. Nous avons ensuite étudié plus précisément les activités de modélisation orientée objet dans le cadre de l'apprentissage de cette discipline et nous avons élaboré un ensemble d'activités jugées pertinentes. Ces activités ont pour but de susciter un apprentissage et de mobiliser différents savoir-faire de la modélisation, en particulier sur le plan métacognitif. Enfin, nous avons étudié et modélisé les rétroactions pertinentes dans le cadre de la modélisation orientée objet et nous avons conçu une taxonomie de rétroactions.

Afin d'éprouver nos études théoriques, nous avons conçu et implémenté dans DIAGRAM plusieurs artefacts informatiques. Nous avons intégré dans cet EIAH un système d'aides métacognitives, et implémenté plusieurs types de tâches. Enfin, nous avons intégré à DIAGRAM un système de rétroactions, sous la forme de messages créés sur la base du résultat de la comparaison du diagramme de l'apprenant avec un diagramme de référence.

Cette comparaison est effectuée par le système de diagnostic dont Ludovic Auxepaules est le concepteur. Enfin, nous avons évalué nos théories et nos productions grâce à plusieurs expérimentations. Ainsi, pour chaque phase de notre travail nous avons observé et évalué la pertinence de nos propositions grâce à des évaluations en contexte écologique.

1.5 Nos propositions

Nos contributions sont à la fois d'ordre théorique et pratique. Du point de vue théorique, nous proposons un modèle d'interaction pour un EIAH pour l'apprentissage de la modélisation. Ce modèle est formé d'un ensemble d'éléments : des modes d'interaction spécifiquement conçus pour l'apprentissage de la modélisation, des aides contextuelles adaptées, une organisation de l'activité particulière et des propositions de rétroactions. Ce modèle a été appliqué au domaine de la modélisation orientée objet.

L'instanciation de notre modèle d'interaction au domaine de la modélisation orientée objet a donné lieu à des réalisations informatiques. Nous avons mis en œuvre nos propositions dans DIAGRAM et nous avons implémenté différents composants informatiques, contribuant ainsi à l'élaboration de cet EIAH.

Enfin nous avons validé nos propositions théoriques et pratiques en menant plusieurs expérimentations de DIAGRAM dans des conditions écologiques d'utilisation.

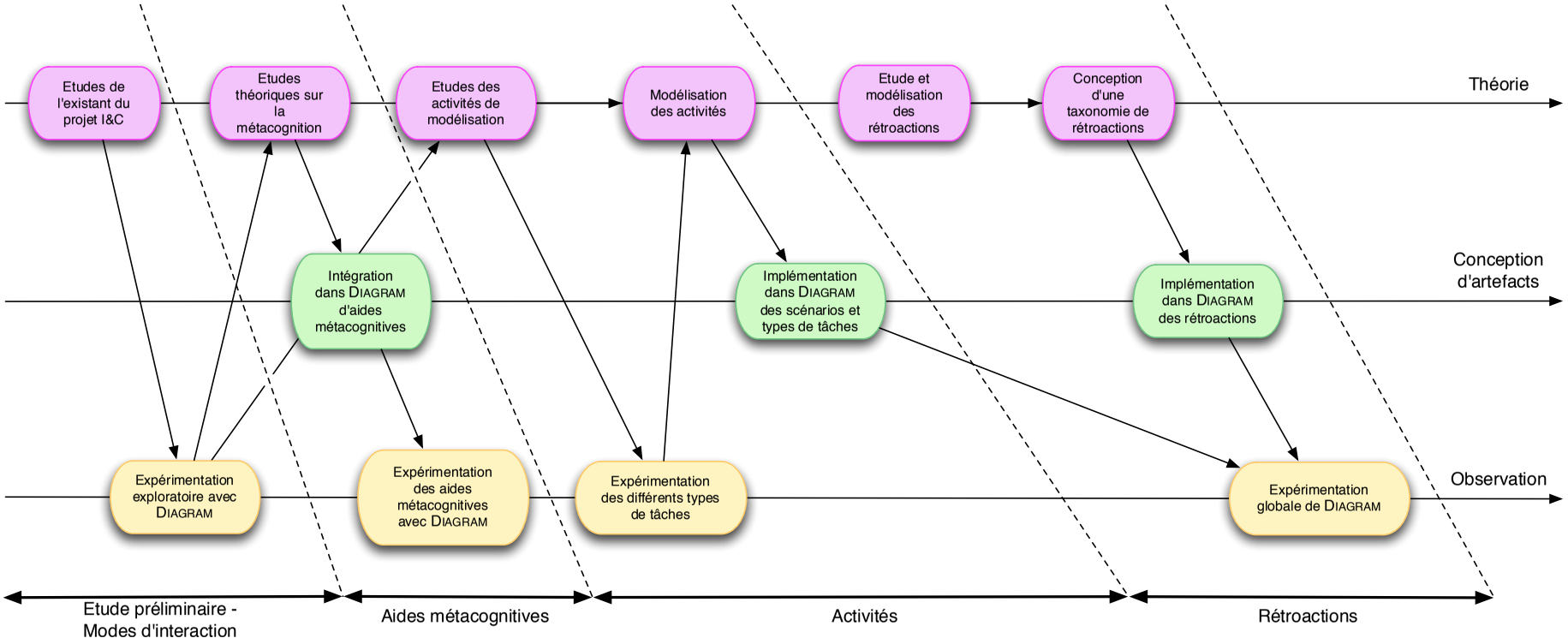


FIG. *1.1 – Méthodologie de recherche*

1.6 Organisation du mémoire

Le deuxième chapitre de ce mémoire est consacré à l'apprentissage de la modélisation orientée objet, en particulier au sein d'environnements informatiques. Nous décrivons les concepts de la modélisation orientée objet utiles à nos travaux puis nous étudions l'apprentissage de cette discipline, les difficultés rencontrées par les apprenants ainsi que les logiciels de modélisation existants. Nous nous intéressons ensuite aux EIAH pour la modélisation en informatique dans le domaine de la modélisation de schémas Entité-Relation et de la modélisation orientée objet.

Nous avons pour objectif d'encourager l'activité métacognitive au cours de l'activité de modélisation. Le troisième chapitre est consacré à l'étude théorique de la métacognition dans les EIAH. Après avoir défini les processus métacognitifs et étudié le rôle de la métacognition pour l'apprentissage, nous décrivons différentes approches et exemples d'intégration de la métacognition dans les EIAH.

Dans le quatrième chapitre, nous présentons notre modèle d'interaction. Il est constitué de quatre composants : les tâches que nous proposons aux apprenants, les modes d'interaction dans l'environnement d'apprentissage, les aides métacognitives et les rétroactions à apporter à l'apprenant. Dans un premier temps nous présentons l'aspect générique de ce modèle, et dans un second temps nous détaillons son application à la modélisation par diagramme de classes qui constitue un aspect essentiel de la modélisation orientée objet.

Le cinquième chapitre présente la mise en œuvre des éléments de notre modèle d'interaction dans DIAGRAM. Dans un premier temps, nous présentons comment chaque élément de notre modèle est implémenté dans cet EIAH. Dans un second temps nous abordons les principes de réalisation informatique de DIAGRAM. Nous présentons l'architecture générale de l'environnement et détaillons certains modules en précisant nos contributions en terme de réalisation informatique.

Le sixième chapitre décrit les différentes expérimentations que nous avons menées. Elles sont de deux types : d'une part des études exploratoires, et d'autre part, des expérimentations permettant de valider nos propositions. Les études exploratoires nous ont permis d'affiner notre modèle. Elles ont consisté en une mise à l'essai d'une version préliminaire de DIAGRAM, et une étude en environnement informatique de nos propositions de tâches d'apprentissage de la modélisation orientée objet. La validation de nos propositions s'est concrétisée par deux expérimentations en contexte écologique. L'une consistait en une validation globale du système DIAGRAM et une analyse de l'effet des rétroactions alors que la seconde se focalisait sur les aides métacognitives.

Nous concluons ce mémoire par un bilan de nos propositions, leurs apports et leurs limites, et nous dégageons des perspectives.

CHAPITRE 2

Apprentissage de la modélisation orientée objet et EIAH

Sommaire

Résumé

Dans ce chapitre, nous nous intéressons à l'apprentissage de la modélisation orientée objet, en particulier au sein d'environnements informatiques. Nous abordons dans un premier temps la notion de modèle informatique et la modélisation des systèmes informatiques selon le paradigme de l'orienté objet puis nous décrivons les principaux éléments du langage UML utiles à notre étude : il s'agit des éléments du diagramme de classes. Nous étudions ensuite l'apprentissage de la modélisation orientée objet, les difficultés rencontrées par les apprenants et les logiciels de modélisation existants. Enfin, nous nous intéressons aux EIAH pour la modélisation en informatique : nous étudions

les logiciels existants dans le domaine des schémas Entité-Relation et de la modélisation orientée objet. Pour terminer, nous positionnons nos travaux par rapport à l'ensemble du chapitre.

2.1 Notion de modèle et modèle informatique

Plusieurs définitions s'appliquent à un modèle d'après le dictionnaire Le Petit Robert [Rey-Debove et Rey, 2002]. Objet d'imitation, il peut servir à la fabrication ou à la reproduction d'un objet. Pour un artiste, c'est l'objet ou la personne dont il reproduit l'image. Un modèle en tant qu'échantillon ou spécimen possède certaines qualités ou caractéristiques qui en font le représentant d'une catégorie. Un modèle est également un objet permettant de reproduire en grande quantité des objets semblables, tel un moule ou un prototype industriel.

[Bézivin et Gerbé, 2001] définissent un modèle comme « une simplification d'un système construit dans une intention particulière. Le modèle doit pouvoir répondre à des questions en lieu et place du système modélisé ».

[Favre et Estublier, 2006] distinguent trois grandes classes de systèmes :

- *Système physique* : système concret et observable comme un animal, le système solaire ou une carte topographique, etc.
- *Système numérique* : système qui réside dans et est manipulé par un ordinateur. C'est à ce genre de système qu'on s'intéresse en informatique. Il peut s'agir par exemple d'un fichier, du contenu d'une base de données ou d'un logiciel, etc.
- *Système abstrait* : système immatériel typiquement manipulé par un cerveau humain comme un compte bancaire, une figure géométrique, les ensembles mathématiques, etc.

Le modèle peut avoir différents usages. Il permet de comprendre, maîtriser et augmenter la fiabilité d'un système. Quand un système est complexe, on peut le décomposer en plusieurs modèles afin d'en représenter tous les aspects. Il peut servir à optimiser l'organisation d'un système afin d'en assurer une meilleure gestion, comme un système informatique de réservation de vols par exemple. Il permet de focaliser sur certains aspects ou besoins spécifiques d'un système sans en connaître tous les détails. Un modèle permet également de simuler un processus. Par exemple, un modèle météorologique sera utilisé pour prévoir les évolutions de phénomènes complexes. Enfin, un modèle peut être utilisé pour tester plusieurs solutions à des problèmes posés en réduisant les coûts et les délais, comme la modélisation d'un futur réseau de transports urbains par exemple.

2.2 Approche orientée objet pour la modélisation des systèmes informatiques

La demande de logiciels sophistiqués alourdit considérablement les contraintes imposées aux équipes de développement. Les professionnels de l'industrie logicielle sont confrontés à une complexité croissante à la fois du fait de la nature des applications, mais aussi des environnements distribués et hétérogènes, de la taille des logiciels, de la composition des équipes de développement, et des attentes des utilisateurs. C'est pour surmonter ces difficultés que les praticiens se focalisent sur la modélisation des logiciels en amont de leur implémentation : c'est l'émergence du génie logiciel qui regroupe

« l'ensemble des activités de conception et de mise en œuvre de produits et procédures tendant à rationaliser la production du logiciel et son suivi ».

Le génie logiciel s'intéresse particulièrement à la manière dont le code source d'un logiciel est spécifié puis produit. Ainsi, il touche au cycle de vie des logiciels qui désigne les différentes phases de développement d'un logiciel. On peut distinguer deux phases majeures du cycle de vie : l'analyse et la conception.

L'analyse consiste à étudier le domaine (contexte, utilisateurs, contraintes, coûts, performances, etc.) dans lequel le système sera intégré, permettant ainsi la conception adéquate de celui-ci. Dans cette étape, on fournit une définition précise du problème à résoudre et on modélise le système pour le rendre plus compréhensible : on structure à l'aide de modèles les informations, les fonctionnalités et le comportement du système. La conception regroupe l'ensemble des techniques et principes appliqués dans le but de définir un système avec suffisamment de détails pour permettre sa réalisation physique.

Ces étapes sont reprises et détaillées dans le cycle de vie des logiciels qui comprend généralement les étapes d'analyse des besoins, de spécification (conception générale), de conception détaillée, d'implémentation, de tests, etc. Plusieurs modèles de cycles de vie existent ainsi que plusieurs méthodes d'analyse et de conception qui fournissent une méthodologie et des notations afin de concevoir des logiciels de qualité. Les deux approches méthodologiques classiques sont l'approche fonctionnelle et l'approche objet. Dans l'approche fonctionnelle, le système est vu comme un ensemble d'unités ayant chacune une fonction clairement définie. L'approche objet s'est imposée plus récemment. Elle est issue du paradigme de programmation orientée objet qui voit le système comme l'assemblage d'objets interagissants.

2.2.1 Approche orientée objet

L'approche orientée objet considère un logiciel comme un ensemble d'objets distincts caractérisés par des propriétés. C'est de l'interaction entre ces objets que naissent les fonctionnalités du logiciel.

A partir des années 80, de nombreuses méthodes de modélisation basées sur les principes de l'orienté objet émergent. La plupart d'entre elles convergent vers des idées communes comme les objets, classes, associations, sous-systèmes, cas d'utilisation. De la prolifération de ces méthodes est né un besoin d'unification et la recherche d'un langage unique utilisable par toutes les méthodes, adapté à toutes les phases du développement et compatible avec toutes les techniques de réalisation. L'unification et la normalisation des trois méthodes dominantes, à savoir OMT (*Object Modeling Technique*) de Rumbaugh, Booch'93 de Booch et OOSE (*Object Oriented Software Engineering*) de Jacobson sont à l'origine de la création du langage UML (*Unified Modeling Language*).

2.2.2 Le langage UML

UML s'est rapidement imposé dans le domaine industriel et la version 1.1 d'UML devient en 1997 un standard de l'OMG (*Object Management Group*). Bien que UML soit issu des méthodes de Rumbaugh, Booch et Jacobson, il ne propose pas une méthodologie complète. Les auteurs d'UML préconisent une démarche pilotée par les cas d'utilisation, centrée sur l'architecture, itérative et incrémentale. Plusieurs processus de développement fondés sur UML existent comme le RUP (*Rational Unified Process*) de Booch, Jacobson et Rumbaugh ou l'approche MDA (*Model Driven Architecture*) proposée par l'OMG mais ils

ne font pas partie du standard UML. UML est donc un langage qui peut être utilisé quelle que soit la méthode suivie.

La version actuelle d'UML 2.1 est constituée de treize types de diagrammes, représentant autant de points de vue d'un système logiciel, et répartis en trois grandes catégories. Six types de diagrammes permettent de représenter l'aspect statique, trois correspondent au comportement du système et quatre représentent les interactions et l'aspect dynamique du système :

- **Diagrammes structurels ou statiques**
  - le diagramme de classes représente la structure statique du système en terme de classes et de relations ;
  - le diagramme d'objets décrit les objets (instances des classes) utilisés dans le système ;
  - le diagramme de composants montre la mise en œuvre physique des composants du système (fichiers, bases de données, etc) ;
  - le diagramme de déploiement représente les éléments matériels et la façon dont les composants du système sont répartis sur ces éléments et interagissent avec eux ;
  - le diagramme de paquetages montre comment les éléments du modèle sont organisés en paquetages et les dépendances entre les paquetages ;
  - le diagramme de structure composite décrit la structure interne des composants d'une classe, les interactions entre les composants eux-mêmes et avec le reste du système.
- **Diagrammes comportementaux**
  - le diagramme des cas d'utilisation représente les fonctions du système du point de vue des utilisateurs ;
  - le diagramme d'états-transitions permet de décrire en terme d'états le comportement du système ou de ses composants ;
  - le diagramme d'activité représente le comportement d'un processus du système ou d'un cas d'utilisation.
- **Diagrammes d'interaction ou dynamiques**
  - le diagramme de séquence est une représentation temporelle des objets et de leurs interactions ;
  - le diagramme de communication est une représentation simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets ;
  - le diagramme global d'interaction permet de décrire les enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquence ;
  - le diagramme de temps décrit les variations d'une donnée au cours du temps.

UML est fondé sur un métamodèle qui décrit formellement la syntaxe, la sémantique et la notation visuelle de chaque élément de modélisation utilisé dans les diagrammes ainsi que les relations entre ces éléments. Le métamodèle décrit également la façon dont les diagrammes peuvent être utilisés, assemblés et enrichis pour modéliser les différents aspects du système étudié. Parmi les diagrammes UML, le diagramme de classes est au centre du processus de modélisation. En phase d'analyse, le diagramme de classes est prévu pour développer la structure des entités manipulées par les utilisateurs. En conception, il représente la structure d'un code orienté objet. C'est le diagramme le plus utilisé et le mieux connu : nous limitons donc notre étude au diagramme de classes.

2.2.3 Le diagramme de classes UML

Le diagramme de classes est un diagramme structurel qui décrit la partie statique d'un système. Il décrit les classes et les relations entre ces dernières, les regroupements de classes en paquetages, les interfaces, les objets et les classes qui participent à une collaboration ou réalisent un cas d'utilisation.

Classe, attribut et opération

Une **classe** est une description d'un ensemble d'objets ayant une sémantique et des caractéristiques en commun. Un objet est une instance d'une classe. Les caractéristiques des classes sont les attributs et les opérations.

Les **attributs** décrivent les informations qu'une classe ou qu'un objet doivent connaître. Ils représentent les données encapsulées dans les objets de la classe. Un attribut est défini par un nom, un type de données et une visibilité. Le nom est donné en analyse mais la spécification de la visibilité et le choix du type sont souvent différés. Un attribut peut également avoir une valeur par défaut. Certains attributs peuvent être calculés à partir d'autres attributs de la classe. Par exemple, l'âge est calculé à partir de la date de naissance d'une personne et de la date actuelle. L'attribut sera noté */age*. Les attributs dérivés sont souvent destinés à devenir des opérations.

Les **opérations** définissent le comportement qu'une classe d'objets peut manifester. Une opération est décrite par son nom, sa visibilité, le type retourné par l'opération, et les arguments qu'elle prend en paramètres. Pour réduire la longueur du libellé, il est possible d'omettre les arguments des opérations, la visibilité et/ou le type retourné.

En UML, les classes sont représentées par des rectangles, divisés en plusieurs compartiments. Dans le premier compartiment figure le nom de la classe, dans le deuxième les attributs et dans le troisième les opérations, comme le montre la figure 2.1.

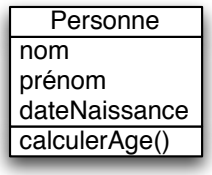


FIG. *2.1 – Représentation UML d'une classe et de ses caractéristiques. Une personne est caractérisée par un nom, un prénom et une date de naissance. On peut calculer son âge en utilisant l'attribut dateNaissance et la date actuelle. Une instance de cette classe est par exemple Jane Doe, née le 01/01/1970.*

Interface

Une interface décrit le comportement visible d'une classe, d'un composant ou d'un sous-système. Ce comportement est défini par une liste d'opérations ayant une visibilité *publique*. Une interface n'a pas d'instances directes. Pour être utilisée, elle doit être réalisée par une classe. Comme le montre la figure 2.2, UML représente les interfaces de deux façons : au moyen d'un petit cercle relié par un trait à l'élément qui fournit les services décrits par l'interface ou comme une classe avec le mot clé **« interface »**. La classe qui réalise cette interface est alors reliée à l'interface par une relation de *réalisation*,

représentée par une flèche en traits pointillés à extrémité triangulaire. Une même classe peut réaliser plusieurs interfaces et une interface peut être réalisée par plusieurs classes.

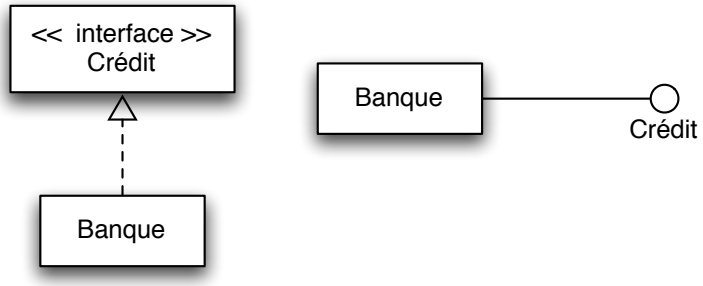


FIG. *2.2 – Représentation UML d'une interface. La classe Banque réalise l'interface Crédit.*

Les classes sont liées entre elles par des **relations** qui expriment des liens sémantiques ou structurels. Même si les relations sont décrites entre les classes, elles expriment souvent des liens entre les objets. Les relations les plus utilisées sont l'association, l'agrégation, la composition, la dépendance et l'héritage.

Association

Une association représente une relation sémantique entre les objets de la classe. Elle est représentée graphiquement par un trait plein entre les classes associées comme le montre la figure 2.3.

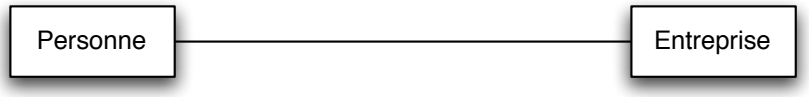


FIG. *2.3 – Représentation UML d'une association entre deux classes*

Nom des associations

L'association est complétée par un nom, qui correspond souvent à un verbe indiquant la nature de la relation entre les classes, avec la précision du sens de lecture en cas d'ambiguïté (cf. figure 2.4).

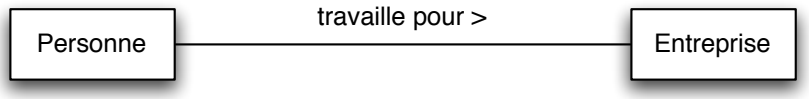


FIG. *2.4 – Représentation UML d'une association nommée. Une personne travaille pour une entreprise.*

Rôle des extrémités des associations

Les rôles des classes dans une relation peuvent être indiqués à chaque extrémité de la relation. Comme le nom des associations, cela permet de clarifier le diagramme.

Multiplicité des associations

Chaque extrémité peut également porter une indication de multiplicité qui spécifie combien d'objets de la classe considérée peuvent être liés à un objet de l'autre classe. Les principales multiplicités normalisées sont décrites dans le tableau 2.1.

TAB. *2.1 – Valeurs de multiplicité principalement utilisées*

Valeurs de multiplicité	**Reformulations**
0..1	Zéro ou un(e)
1	Un(e) unique
0..* ou *	Zéro ou plusieurs
1..*	Au moins un(e)
N	Exactement N (entier naturel)
N..M	Entre N et M (entiers naturels)

La figure 2.5 montre un exemple d'association nommée, comprenant les rôles et les multiplicités des extrémités.

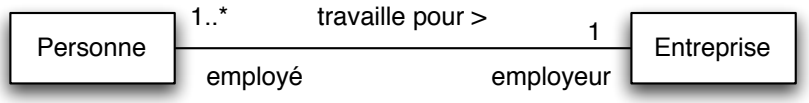


FIG. *2.5 – Représentation UML complète d'une association entre deux classes. Une personne travaille pour une et une seule entreprise. L'entreprise emploie au moins une personne. L'entreprise est l'employeur des personnes qui travaillent pour elle et une personne a un statut d'employé dans l'entreprise.*

Association n-aire

La plupart des associations sont binaires (elles relient deux classes uniquement). Des arités supérieures peuvent cependant exister et se représentent alors au moyen d'un losange sur lequel convergent les différentes composantes de l'association (cf. figure 2.6).

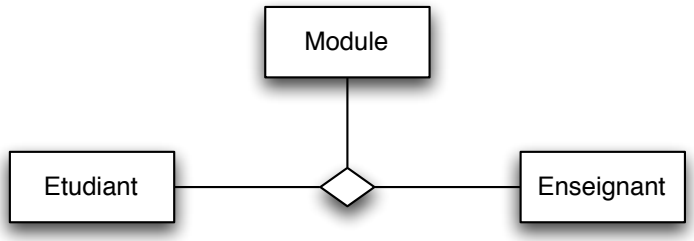


FIG. *2.6 – Représentation UML d'une association ternaire. Un étudiant suit des modules assurés par des enseignants.*

Classe-association

Une association peut être raffinée et avoir ses propres propriétés, qui ne sont disponibles dans aucune des classes qu'elle lie. Comme seules les classes peuvent avoir des propriétés,

cette association devient alors une classe appelée **classe-association** (cf. figure 2.7).

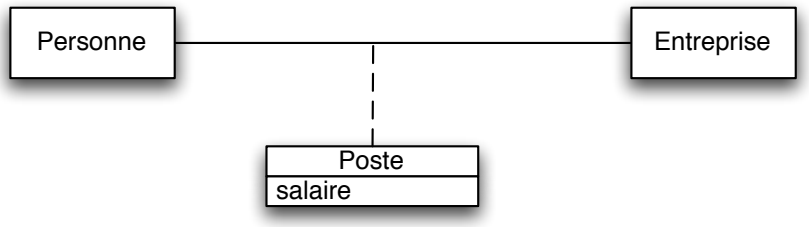


FIG. *2.7 – Représentation UML d'une classe-association. Les personnes qui travaillent dans une entreprise occupent un poste. Parmi les propriétés associées au poste figure le salaire.*

Agrégation

Une agrégation est une association particulière. Elle représente la relation d'inclusion structurelle ou comportementale d'un élément dans un ensemble. Dans la notation graphique d'UML, elle se distingue d'une association par l'ajout d'un losange vide du côté de l'agrégat (cf. figure 2.8). La durée de vie de l'agrégat est indépendante des éléments qui le constituent : dans notre exemple, la destruction d'une instance de la classe 'Personne' n'implique pas la destruction des instances de la classe 'Immeuble'. Par ailleurs, un composant peut apparaître dans plusieurs agrégats : un immeuble peut appartenir à un ensemble de copropriétaires.

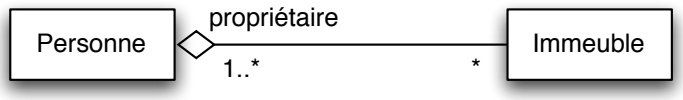


FIG. *2.8 – Représentation UML d'une agrégation. Des personnes peuvent être copropriétaires d'un même immeuble sur lequel elles possèdent des droits conjoints.*

Composition

Une composition est une agrégation plus forte impliquant que la destruction de l'objet composite entraîne la destruction de ses composants. De plus une instance d'un composant appartient toujours à une instance de l'élément composite au plus. Une composition se distingue d'une association par l'ajout d'un losange plein du côté du composite (cf. figure 2.9).

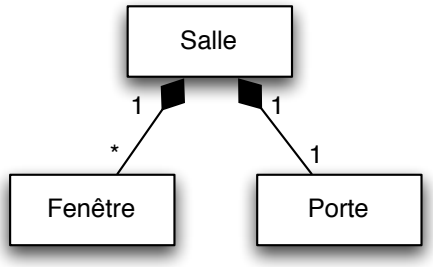


FIG. *2.9 – Représentation UML d'une composition. Une salle est composée d'une seule porte et de plusieurs fenêtres.*

Dépendance

Une dépendance est une relation unidirectionnelle utilisée lorsqu'il existe une relation sémantique entre plusieurs éléments du modèle qui n'est pas de nature structurelle. Elle indique que la modification de la cible implique le changement de la source. La relation de dépendance est représentée par une flèche en traits discontinus pointant vers l'élément cible (cf. figure 2.10).

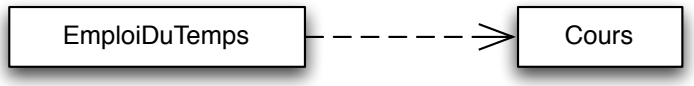


FIG. *2.10 – Représentation UML d'une relation de dépendance. Le déroulement des cours dépend de l'emploi du temps.*

Héritage

Un des intérêts de la modélisation orientée objet est la possibilité de manipuler des concepts abstraits. Ce principe d'abstraction est assuré par le concept d'**héritage** qui permet de simplifier le modèle en factorisant ses propriétés. L'abstraction consiste à définir une hiérarchie entre des éléments ayant des propriétés communes. Les propriétés communes sont rassemblées dans une super-classe, ou classe-parent, par le processus de *généralisation* et les propriétés spécifiques restent dans les sous-classes, ou classes-enfants, par le processus de *spécification*. L'héritage est représenté en UML au moyen d'une flèche orientée de la sous-classe vers la super-classe (cf. figure 2.11).

La liste suivante donne quelques propriétés de l'héritage :

- La classe-enfant possède toutes les propriétés de ses classes-parents mais elle n'a pas accès aux propriétés privées de celles-ci.
- Une classe-enfant peut redéfinir une ou plusieurs opérations de la classe-parent. Sauf indication contraire, un objet utilise les opérations les plus spécialisées dans la hiérarchie des classes. La surcharge d'opérations (même nom, mais signature des opérations différentes) est possible dans toutes les classes.
- Toutes les associations de la classe-parent s'appliquent par défaut à ses sous-classes.
- Une instance d'une classe peut être utilisée partout où une instance de sa classe-parent est attendue (principe de la substitution). Par exemple, toute opération acceptant un objet d'une classe *FigureGéométrique* doit accepter tout objet de la classe *Rectangle* (l'inverse n'est pas toujours vrai).

– Une classe peut avoir plusieurs classes-parents : on parle alors d'héritage multiple.

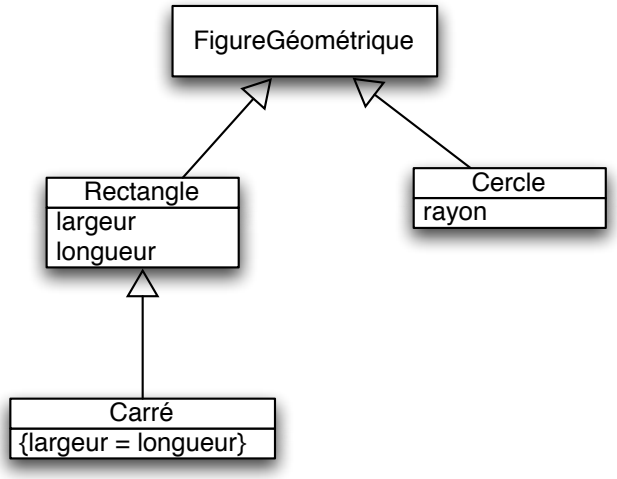


FIG. *2.11 – Représentation UML d'une relation d'héritage. Un rectangle et un cercle sont des figures géométriques. Elles spécialisent la classe FigureGéométrique en ajoutant des attributs. La classe Carré spécialise la classe Rectangle en réduisant le domaine de définition : un carré est un rectangle dont la longueur et la largeur sont identiques.*

Classe abstraite

Les classes abstraites ne sont pas instanciables directement : elles servent de spécification plus générale pour manipuler les objets instances d'une ou de plusieurs de leurs sous-classes. Par convention, le nom des classes abstraites est écrit en italique. Une classe abstraite se différencie d'une interface puisqu'elle peut posséder des attributs et que ses opérations peuvent posséder des implémentations. Une classe est abstraite lorsque au moins une des opérations qu'elle définit ou dont elle hérite est abstraite, c'est-à-dire qu'elle ne possède pas d'implémentation. La figure 2.12 montre comment le procédé d'abstraction permet de compléter l'exemple précédent.

Principe d'encapsulation

Le principe d'encapsulation définit les droits d'accès aux propriétés des éléments du système. Une propriété peut être visible partout : dans ce cas, elle est dite *publique*. Si elle n'est visible qu'à l'intérieur d'une classe, elle est dite *privée*. En utilisant le principe de l'héritage, les descendants d'une classe peuvent avoir un accès privilégié aux propriétés des classes-parents. Une classe-parent qui souhaite autoriser l'accès à une de ses propriétés à ses seuls descendants doit définir cette propriété comme *protégée*. Lors de la modélisation d'applications complexes, les classes sont regroupées dans des paquetages dédiés à des domaines ou applications particulières. La visibilité par défaut d'une classe et de ses propriétés se limite au paquetage dans lequel elle est définie.

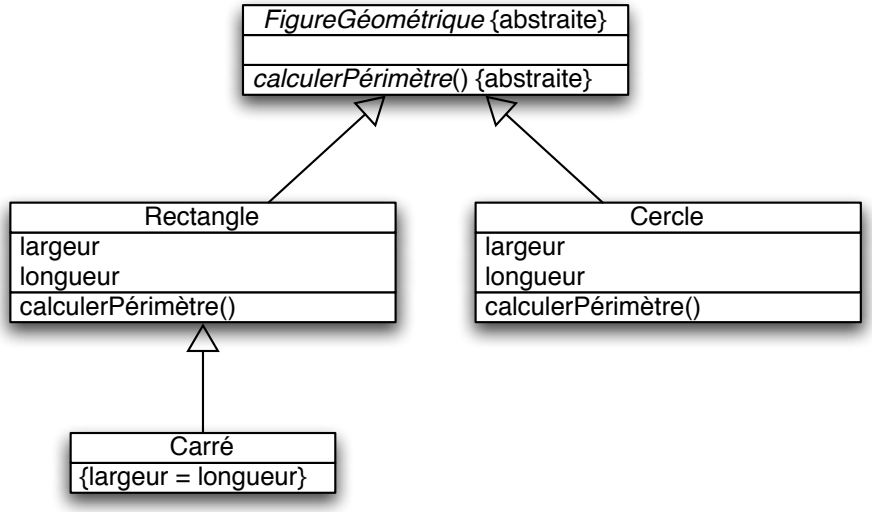


FIG. *2.12 – Représentation UML d'une classe abstraite. La classe abstraite FigureGéométrique définit l'opération abstraite calculerPérimètre dont l'implémentation diffère selon les sous-classes Rectangle et Cercle. La classe Carré ne redéfinit pas l'opération calculerPérimètre : c'est l'implémentation de la classe Rectangle qui sera utilisée.*

2.3 Apprentissage de la modélisation orientée objet

Depuis l'émergence du génie logiciel et la standardisation d'UML, la modélisation orientée objet est intégrée aux enseignements des filières universitaires et professionnalisantes en informatique et de nombreux ouvrages consacrés à la modélisation orientée objet et au langage UML ont vu le jour. Nous nous intéressons dans ce paragraphe à l'apprentissage de la modélisation orientée objet et en particulier à l'apprentissage de la modélisation par diagramme de classes UML. Pour cela, nous étudions dans un premier temps l'initiation à la modélisation orientée objet puis nous identifions les difficultés de cet apprentissage. Enfin, nous nous intéressons aux logiciels de modélisation UML existants.

2.3.1 Initiation à la modélisation orientée objet

Notre étude de l'initiation à la modélisation orientée objet s'est effectuée selon deux axes. Le premier consiste en l'étude d'un ensemble d'ouvrages consacrés à la modélisation orientée objet qui font référence dans le domaine. L'analyse de ces ouvrages nous a permis de repérer les pratiques et les conseils de modélisation donnés aux lecteurs. Le second consiste en l'étude de l'apprentissage de la modélisation orientée objet en formation initiale.

2.3.1.1 Ouvrages dédiés à la modélisation orientée objet

Les objectifs des ouvrages dédiés à la modélisation orientée objet sont divers : certains sont exhaustifs sur la notation UML, d'autres se veulent une référence concise du langage. Tous contiennent une étude de cas dont certains font le fil rouge de l'ouvrage.

Nous avons principalement étudié les ouvrages suivants : [Penders, 2002], [Charroux et al., 2005], [Roques, 2003], [Roques et Vallée, 2007] et

[Muller et Gaertner, 2000] afin de repérer les pratiques et conseils méthodologiques pouvant faciliter la modélisation par diagramme de classes par des novices.

Dans les ouvrages de [Muller et Gaertner, 2000] et [Roques et Vallée, 2007], la réalisation du diagramme de classes est replacée dans le processus global de modélisation d'un système en vue de sa conception. Une ébauche du diagramme de classes est réalisée au cours de la description des cas d'utilisation. Cette ébauche est ensuite affinée et complétée tout au long de l'étape d'analyse. [Roques et Vallée, 2007] soulignent l'importance d'affiner les classes identifiées et les associations, et donnent des principes généraux et des règles pour y parvenir. Ces deux ouvrages sont basés sur une étude de cas. [Muller et Gaertner, 2000] détaillent de façon exhaustive la notation UML et proposent une démarche simple et générique de modélisation avec UML ainsi que deux exemples de mise en œuvre. [Roques et Vallée, 2007] se veulent un guide pratique d'utilisation d'UML et proposent une étude de cas qui couvre toutes les étapes du développement. Par exemple, ils précisent que les « premières classes candidates identifiées doivent être des concepts connus des utilisateurs du système » et qu'il faut chercher « les noms communs importants dans les descriptions textuelles des cas d'utilisation ». Il faut ensuite examiner les classes candidates, en éliminer certaines ou en ajouter d'autres. Il faut également affiner les associations, ajouter des attributs et des opérations au cours d'un processus itératif de validation du modèle.

A destination des programmeurs, chefs de projet et analystes clients, [Penders, 2002] propose les concepts fondamentaux de la modélisation avec UML, la description des diagrammes UML et leur application à une étude de cas. Concernant le diagramme de classes, il est construit dans l'étude de cas à partir d'une description textuelle du problème. Les étapes de construction proposées sont les suivantes :

- identifier les classes à partir de la description textuelle du problème ;
- identifier et nommer les associations entre paires de classes. Affecter les multiplicités et contraintes si nécessaire ;
- évaluer chaque association pour déterminer si elle devrait être une agrégation, et si c'est le cas, vérifier s'il peut s'agir d'une composition ;
- évaluer les classes afin de savoir si une spécialisation ou une généralisation sont possibles.

Pour chaque problème, un diagramme de classes est proposé et discuté en relation avec le texte du problème. L'auteur conseille d'être attentif au vocabulaire de la description du problème.

Ces conseils sont également présents dans l'ouvrage de [Charroux et al., 2005], pour qui une démarche couramment utilisée pour bâtir un diagramme de classes consiste à :

- trouver les classes qui correspondent souvent à des concepts ou à des substantifs du domaine ;
- trouver les associations entre les classes. Les associations correspondent généralement à des verbes ;
- trouver les attributs des classes qui sont souvent des substantifs tels que « la masse d'une voiture » ou « le montant d'une transaction » ;
- organiser et simplifier le diagramme en utilisant le principe de généralisation.

Les auteurs précisent également qu'il faut réitérer ce processus afin d'affiner et d'améliorer la qualité du modèle.

L'ouvrage de [Roques, 2003] est à visée pédagogique. Il propose une base d'exercices corrigés permettant de se familiariser avec les différents points de vue de la modélisation d'un système. Pour chaque point de vue il propose une application à une étude de cas,

des exercices supplémentaires et des conseils. Concernant le point de vue statique et la modélisation du diagramme de classes, l'auteur propose de se baser sur des interviews d'experts résumant leur connaissance du domaine sous forme de phrases. Chaque phrase est analysée et permet de créer, compléter et affiner le diagramme de classes.

Les ouvrages étudiés abordent la modélisation du diagramme de classes dans deux situations différentes : le cadre professionnel et le cadre pédagogique. Dans un contexte professionnel, le diagramme de classes est réalisé à partir des concepts identifiés dans les cas d'utilisation. Dans une situation d'apprentissage, les auteurs se basent sur une description textuelle du problème à modéliser. Comme nous l'avons évoqué dans le paragraphe 2.2.2, UML ne fournit pas de méthodologie de modélisation. En cela, il n'y a pas de démarche précise pour concevoir le diagramme de classes d'un système mais les ouvrages que nous avons étudiés proposent plusieurs éléments méthodologiques pour le construire, tels que le repérage des noms et des verbes pour identifier les classes, attributs et relations du diagramme. Enfin, tous les ouvrages soulignent l'importance de l'aspect incrémental du processus.

2.3.1.2 Apprentissage de la modélisation orientée objet en formation initiale

Nous avons étudié plusieurs formations et supports d'enseignement de la modélisation avec UML. A l'université du Maine, nous avons étudié les enseignements de modélisation orientée objet de seconde année de DEUST (Diplôme d'Etudes Universitaires Scientifiques et Techniques) ISR (Informatique, Systèmes et Réseaux), une formation professionnalisante en informatique sur deux ans, et ceux donnés en première année de DUT (Diplôme Universitaire de Technologie) Informatique de Laval. Les autres supports que nous avons étudiés proviennent d'Internet. Il s'agit de supports mis à disposition par des enseignants du domaine, et de natures diverses : transparents de cours sur les diagrammes de classes, exemples de modélisation à partir de textes, cas d'étude ou sujets d'examens.

De manière générale, l'enseignement de la modélisation orientée objet en formation initiale repose d'une part sur des cours magistraux visant à présenter les connaissances théoriques (syntaxe et sémantique du langage UML) et d'autre part sur des travaux dirigés (TD) et pratiques (TP) ayant pour objectif de faire acquérir le savoir-faire de modélisation, par la pratique, à l'aide d'exercices de complexité croissante. Les exercices d'apprentissage sont généralement des descriptions de problèmes à modéliser ou des études de cas telles qu'on en trouve dans les ouvrages consacrés à la modélisation avec UML. Les modalités d'actions sont différentes en TD et en TP : les TD se déroulent sur papier, alors que les TP sont l'occasion d'utiliser les logiciels d'édition de diagrammes UML.

2.3.2 Difficultés de l'apprentissage de la modélisation orientée objet

Construire un modèle n'est pas un problème à solution unique et dans de nombreux cas, le même problème analysé par des personnes différentes aboutit à des modèles différents. Ces variations correspondent généralement à des différences de point de vue. Ainsi, il n'y a pas de bon ou de mauvais modèle, en fonction des objectifs et des aspects considérés du système, certains modèles sont plus adaptés, ou plus génériques que d'autres. Les auteurs de [Habra et Noben, 2001] décrivent l'enseignement de la modélisation orientée objet en amont de la programmation, à l'université de Namur. Dans leur étude, ils constatent que les principaux problèmes rencontrés au cours de la modélisation orientée objet par les

apprenants sont dus à un manque d'expérience. Ils soulignent le fait qu'aucune règle universelle ne s'applique pour la création d'un modèle et que la créativité a une place prépondérante dans le processus de modélisation. Les auteurs précisent que les apprenants ont tout de même besoin d'un ensemble de conseils méthodologiques à suivre pour diriger leur créativité. Ce point de vue est partagé par [Frosch-Wilke, 2003] pour qui les apprenants ont besoin de processus ou d'étapes à suivre pour construire leur diagrammes. Selon lui, la modélisation s'acquiert par la pratique et les enseignements d'UML doivent s'intégrer dans des projets réalistes pour que les apprenants mesurent l'intérêt de l'utilisation de ce langage.

Ces constatations montrent que les connaissances théoriques des concepts du langage UML ne suffisent pas pour s'approprier la modélisation orientée objet : des capacités de haut niveau comme l'analyse du problème et l'abstraction sont indispensables et s'acquièrent par la pratique.

Dans son étude didactique d'un cours de génie logiciel basé sur l'approche orientée objet, [Surcin et al., 1995] identifient les écueuils à éviter lors de l'étape d'écriture des spécifications logicielles en phase d'analyse. Parmi les écueuils figurent l'*incomplétude*, l'*inconsistance* et l'expression d'une solution plutôt qu'une description du problème posé. Ce dernier cas relève d'un mauvais niveau d'*abstraction*. Les auteurs identifient la notion d'abstraction comme une notion fondamentale du cours de génie logiciel mais ils la considèrent comme difficile à enseigner.

[Holland et al., 1997] identifient et décrivent un certain nombre de conceptions incorrectes observées chez les apprenants durant l'apprentissage de l'approche orientée objet. Par exemple, la confusion entre objet et classe, ou entre attribut et identité de l'objet, en particulier entre un attribut 'nom' d'une classe 'compte' et le nom identifiant un objet de cette classe ('monCompte' par exemple).

Enfin, une difficulté de l'apprentissage de la modélisation provient des logiciels de modélisation utilisés au cours des travaux pratiques [Gayler et al., 2007]. Ces éditeurs sont la plupart du temps des logiciels professionnels qui ne sont pas adaptés aux finalités pédagogiques de l'enseignement : ils sont généralement complexes, longs à maîtriser et présentent des fonctionnalités inutiles au cours des premières séances d'apprentissage.

Dans le domaine de la modélisation de schémas Entité-Relation, les mêmes difficultés sont constatées par [Hall et Gordon, 1998] et [Ghandeharizadeh, 2003].

2.3.3 Logiciels de modélisation UML

Il existe de nombreux logiciels permettant de construire des diagrammes dans le formalisme UML. Nous avons laissé de côté les éditeurs graphiques « généraux » pour concentrer notre étude sur les logiciels dédiés à UML. Nous avons classé ces derniers en deux catégories : les outils destinés à un usage professionnel et les outils dont les auteurs revendiquent un usage dans un contexte d'apprentissage.

2.3.3.1 Outils professionnels

Le marché des outils consacrés à la modélisation avec UML a explosé. Des produits gratuits et *open-source* aux systèmes haut de gamme, il existe des dizaines d'éditeurs UML. [Penders, 2002] propose un ensemble de critères d'évaluation de ces outils. Parmi les fonctionnalités de base figurent le respect d'une norme UML donnée (s'assurant donc que tous les diagrammes peuvent être construits en respectant cette norme), l'environnement supporté, la possibilité d'imprimer les diagrammes, la documentation, le partage des

ressources (lors d'un projet en équipe par exemple), la génération de code, l'éditeur intégré, le contrôle de version. Parmi les fonctionnalités étendues, on trouve l'ingénierie circulaire (maintenance du diagramme par le code et inversement), la modélisation intégrée des données, la personnalisation, l'échange de métadonnées XML ou le travail en collaboration. Ces critères sont adaptés à une utilisation professionnelle du logiciel mais de nombreuses fonctionnalités sont inutiles pour une initiation à la modélisation orientée objet. Elles se retrouvent à divers degrés dans les logiciels dédiés à la modélisation avec UML.

Parmi les logiciels dédiés à la modélisation UML, nous trouvons de nombreux outils commerciaux, très complets et sophistiqués ayant pour vocation une utilisation professionnelle, comme par exemple : **MagicDraw** [MagicDraw], **IBM RationalRose** [RationalRose], **Visual Paradigm** [VisualParadigm], **Jude** [Jude], **Together** [Together], **Objecteering** [Objecteering], **Poseidon** [Poseidon] ou **UMLStudio** [UMLStudio].

Il existe également de nombreux logiciels libres. Au sein de cet ensemble, nous trouvons de nombreux *plugins* intégrables à l'environnement de développement Eclipse comme **Papyrus** [Papyrus], **Omondo** [Omondo], **AgileJ** [AgileJ] ou l'environnement **Fujaba LIFE**[3] [Fujaba, Meyer et Wendehals, 2004]. L'utilisation d'un *plugin* Eclipse pour la modélisation des diagrammes vise plutôt les développeurs en Java que les apprenants en modélisation.

D'autres logiciels ont été développés pour un environnement graphique particulier ou une plateforme spécifique. Nous pouvons citer par exemple **Gaphor** [Gaphor], un éditeur développé en Python pour la plateforme GNOME ; **Umbrello** [Umbrello] un outil développé pour l'environnement KDE et les distributions GNU/Linux ; **Bouml** [Bouml], un environnement multi-plateformes dont la spécificité est la rapidité d'exécution et **Astade** [Astade] qui est orienté vers la conception d'un modèle UML en vue de la génération du code en C++.

2.3.3.2 Outils à visée éducative

De nombreux travaux portent sur la conception d'outils éducatifs pour l'apprentissage du paradigme orienté objet, en amont de la programmation orientée objet ou en lien avec elle. **BlueJ**, par exemple, est un environnement de développement intégré (IDE) développé spécifiquement pour l'apprentissage de la programmation orientée objet [Kölling et al., 2003]. L'interface très simplifiée de BlueJ permet de visualiser et manipuler le diagramme de classes UML de l'application construite. Les apprenants peuvent créer des classes et des objets, ajouter et appeler des opérations. Ils construisent progressivement un projet en entier.

Les auteurs de [Crahen et al., 2002, Alphonce et Ventura, 2003] partent du constat que les apprenants ne saisissent pas toujours l'importance de la conception en amont de la programmation. Ils proposent donc **QuickUML**, un outil écrit en Java permettant la création de diagrammes de classes UML et la génération de code Java mais également la rétroconception à partir du code source Java. QuickUML a été réalisé dans le but d'être utilisé par des apprenants grâce à une interface simplifiée qui propose un sous-ensemble des éléments UML utiles au cours de l'enseignement.

Green est un *plugin* Eclipse pour l'ingénierie et la réingénierie qui est à destination des apprenants [Green, Alphonce et Martin, 2005]. Il permet aux apprenants de se concentrer sur la modélisation plutôt que sur des détails syntaxiques. Green donne la possibilité aux apprenants de se déplacer d'un haut niveau de conception à un code de bas niveau à

tout moment : la vue du diagramme de classes et la vue du code sont simplement des perspectives différentes d'un même artefact. Selon les auteurs, la compréhension de la sémantique véhiculée par le modèle est facilitée par les liens entre le modèle et le code correspondant.

MinimUML [MinimUML, Turner et al., 2005], comme QuickUML, propose uniquement un sous-ensemble d'éléments UML jugés utiles pour une introduction à la modélisation orienté objet, et ne permet que la création de diagrammes de classes. Les diagrammes contiennent deux éléments de base : les classes et les relations, traitées de façon générique. Le choix du type de classe (classe ou interface) et du type de relation (association, agrégation ou héritage) se fait après la création. L'outil propose de nombreuses fonctionnalités graphiques (zoom, copié-collé) ainsi que la génération de code élémentaire pour Java et C++.

UMLet [UMLet] est un éditeur UML open-source permettant de construire rapidement des diagrammes et de les exporter en image. L'interface comprend un espace d'édition, une palette d'éléments en fonction du diagramme choisi, et une fenêtre de propriétés (cf. figure 2.13). Lorsqu'un élément du diagramme est sélectionné, ses propriétés s'affichent et peuvent être modifiées dans cette fenêtre. Multi-plateformes, il fonctionne seul ou comme un *plugin* Eclipse. UMLet est un outil utile pour dessiner rapidement des diagrammes. En revanche, il n'intègre pas de sémantique UML ni d'aides particulières pour créer ces diagrammes.

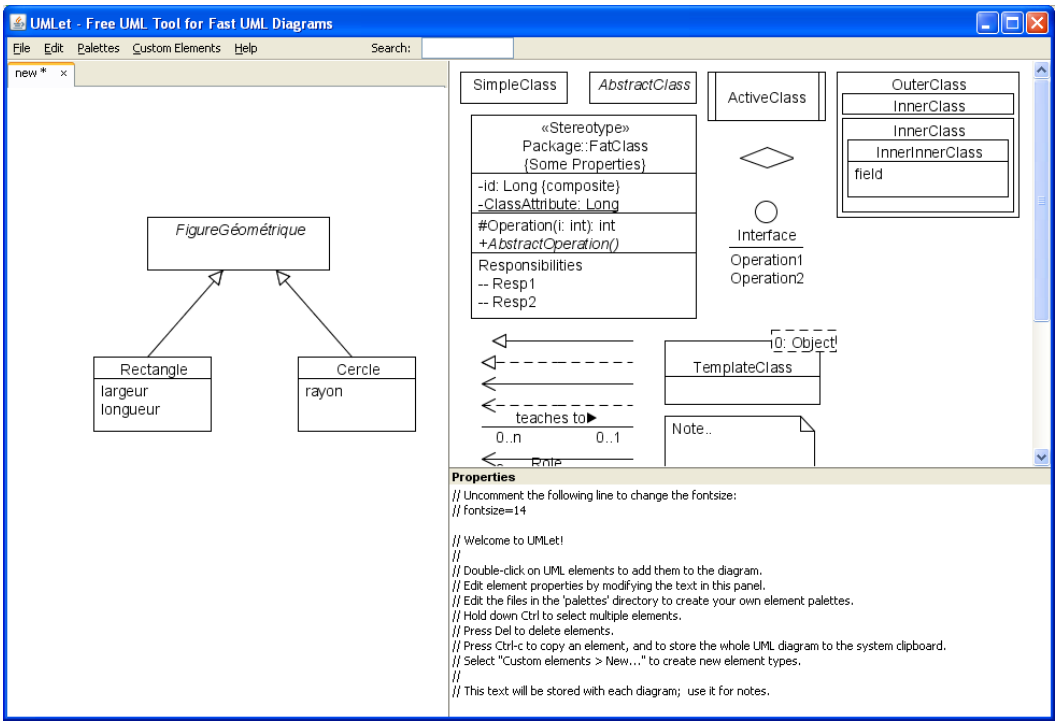


FIG. *2.13 – Interface de UMLet*

Violet [Violet] est également un éditeur UML libre qui peut être lancé indépendamment ou intégré dans Eclipse. Il propose les modèles de diagrammes de cas d'utilisation, de classes, d'objets, d'états, d'activité et de séquence. Les auteurs revendiquent la nécessité d'un éditeur plus simple que les outils professionnels pour un public d'étudiants, enseignants ou auteurs de diagrammes qui ont besoin de créer rapidement des diagrammes UML simples, dans un cadre non professionnel. Violet propose donc des fonctionnalités

telles que le zoom, l'exportation en image ou l'impression mais n'intègre pas de vérification du diagramme. L'interface de Violet contient l'espace de modélisation, les boutons standards d'édition (zoom, copie, suppression, etc.) et les éléments de modélisation du diagramme choisi (nom et représentation graphique de chaque élément) comme le montre la figure 2.14.

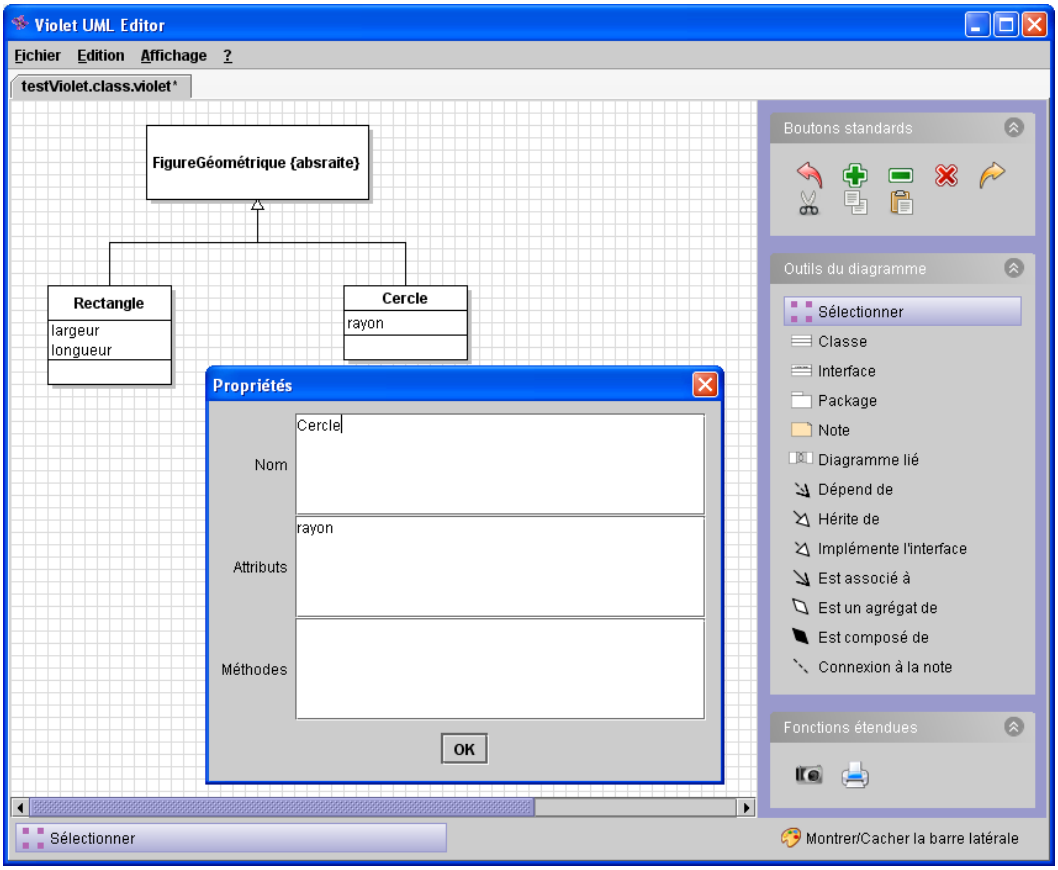


FIG. *2.14 – Interface de Violet. Edition des propriétés de la classe Cercle.*

ArgoUML [ArgoUML] est un logiciel libre à l'origine du produit commercial Poseidon. Développé en Java, il permet la création de diagrammes UML et la génération de code Java, C++ ou Php. Cet outil est issu des travaux de recherche de [Robbins, 1999] qui a développé une librairie de fonctions de soutien cognitif mise en œuvre dans quatre outils, dont ArgoUML. Les fonctions cognitives qu'ArgoUML propose sont par exemple l'affichage de différentes listes de remarques comme des éléments à vérifier ou à compléter pour améliorer le modèle. On peut afficher ces listes selon la priorité des remarques (élevée, moyenne, basse), par sujet (les relations, l'affectation de nom, etc.), ou par type (sémantique, syntaxique, optimisation, etc). ArgoUML propose également un mécanisme de liste de contrôle sensible au contexte : en fonction de l'élement sélectionné, cette liste contient un ensemble de questions que le concepteur du modèle doit se poser pour éviter les erreurs et améliorer son diagramme comme le montre la figure 2.15. Les fonctionnalités de ArgoUML en font un outil particulièrement intéressant dans un contexte d'apprentissage.

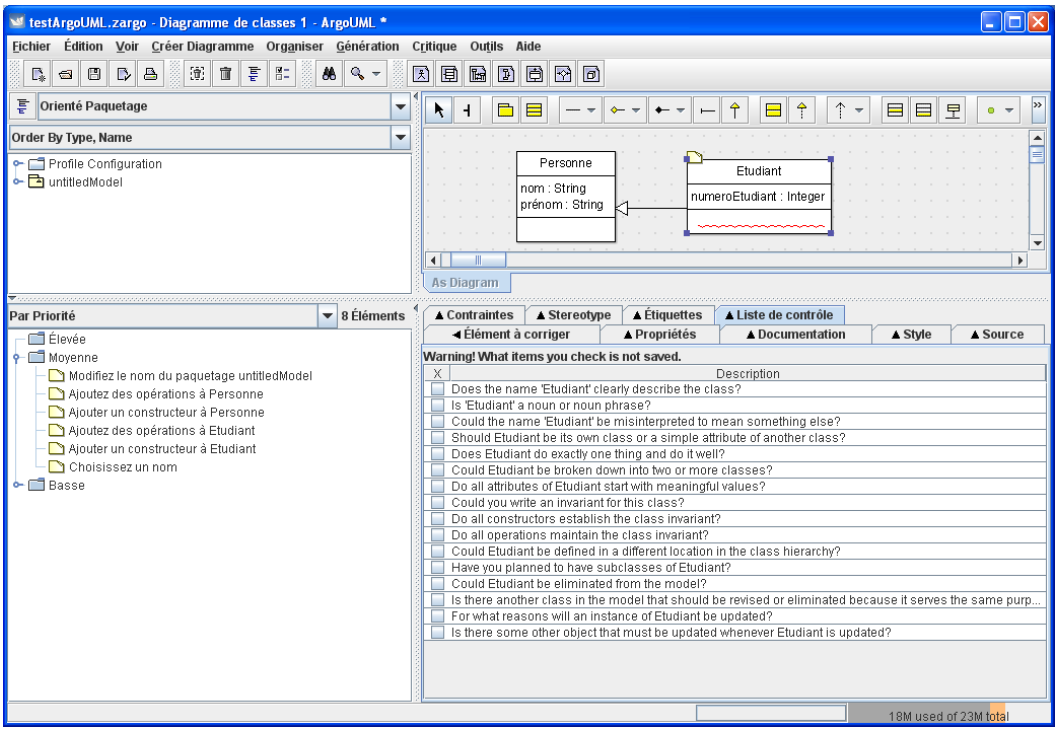


FIG. *2.15 – Interface de ArgoUML. En cliquant sur la classe nommée 'Etudiant', nous obtenons par exemple les questions suivantes : Est-ce que le nom 'Etudiant' décrit clairement la classe ? Est-ce que le nom 'Etudiant' ne peut pas être mal interprété ? Est-ce que la classe 'Etudiant' ne peut pas être éliminée du modèle ?*

StudentUML est un outil à destination des apprenants pour la modélisation UML [Ramollari et Dranidis, 2007, Dranidis, 2007]. StudentUML se focalise sur la vérification de la correction des diagrammes construits et sur la cohérence entre les diagrammes d'un même projet. Au cours de la modélisation, si une erreur de cohérence est détectée, l'apprenant reçoit un message d'erreur ou un avertissement. Il a la possibilité de laisser le système résoudre automatiquement le problème détecté.

Enfin, certains outils proposent l'utilisation d'interfaces originales pour l'apprentissage de la modélisation avec UML. **IdeogramicUML** est un outil commercial [IdeogramicUML, Hansen et Ratzer, 2002] basé sur l'utilisation d'un tableau blanc pour construire le diagramme en collaboration. Les utilisateurs dessinent des formes qui sont ensuite transformées en éléments UML : par exemple, le dessin d'une forme rectangulaire permet la création d'une classe.

SketchUML [SketchUML, Tenbergen et al., 2008] est un outil pour l'apprentissage de la syntaxe UML développé pour TabletPC. Comme IdeogramicUML, il est basé sur la reconnaissance des formes tracées dans une interface épurée, proche d'un environnement papier-crayon traditionnel. Il propose un mode enseignant, dans lequel l'enseignant peut critiquer le diagramme de l'apprenant à l'aide d'une encre rouge. Les éléments ajoutés, supprimés ou corrigés du diagramme seront mis en évidence pour que l'apprenant repère aisément les modifications apportées par l'enseignant.

2.4 Approches en EIAH pour l'apprentissage de la modélisation en Informatique

Dans ce paragraphe, nous présentons quelques exemples d'EIAH pour la modélisation informatique dans le domaine des schémas Entité-Relation et de la modélisation orientée objet.

2.4.1 EIAH pour la modélisation de schémas Entité-Relation

2.4.1.1 CODASYS

CODASYS (Consulting system for Database Design) est un système dont l'objectif est d'assister les concepteurs novices au cours de leur modélisation d'un schéma Entité-Relation [Antony et Batra, 2002]. C'est un outil générique (il ne contient pas de connaissances du domaine) basé sur l'approche de prévention des erreurs. CODASYS n'autorise que la création de modèles corrects selon un certains nombre de règles établies comme le respect de la quatrième forme normale, l'unicité des noms des entités et des attributs, etc. Au cours de la modélisation, le système assiste l'apprenant en posant des questions pour l'aider à compléter son modèle. Par exemple, lorsque l'apprenant modélise une relation 'Travaille' entre les entités 'Employé' et 'Département', le système affiche la question suivante : « Etant donné une instance de 'Employé', combien d'instances de 'Département' peuvent être engagées dans la relation 'Travaille' ? ». L'apprenant peut ainsi répondre *un* ou *plusieurs*.

L'évaluation empirique de CODASYS a montré que les modèles réalisés par les utilisateurs étaient « meilleurs » que les modèles réalisés par le groupe de contrôle, utilisant un autre outil.

2.4.1.2 ERM-VLE

ERM-VLE (Entity Relationship Modelling Virtual Learning Environment) [Hall et Gordon, 1998] est un environnement virtuel d'apprentissage de la modélisation de schémas Entité-Relation dans lequel les apprenants construisent les bases de données en naviguant dans un monde virtuel et en manipulant des objets de ce monde. Le monde virtuel est constitué de différentes pièces comme la pièce de création d'entités ou la pièce de création de relations. D'après les auteurs, cette organisation de l'environnement reflète la structure de la tâche de modélisation. Les apprenants manipulent les objets du monde virtuel grâce à des commandes textuelles comme *ramasser, laisser tomber, nommer, évaluer, créer, supprimer*. L'effet de la commande dépend de l'endroit où elle est effectuée. Par exemple, l'apprenant crée une entité lorsqu'il se trouve dans la pièce de création d'entités. La commande d'évaluation donne des indices sur la modélisation du schéma Entité-Relation. L'interface de ERM-VLE est constituée de plusieurs volets. Le volet *Scénario* contient la description textuelle des besoins pour la base de données modélisée dans l'exercice. Le volet *Current ERM* fournit une représentation graphique du modèle construit par l'apprenant. Cette représentation graphique est dynamiquement mise à jour en fonction de l'activité de l'apprenant mais elle n'est pas manipulable directement. L'apprenant ne peut interagir avec le monde virtuel que par le biais des commandes textuelles. Le volet *ERM World* contient l'historique des interactions entre l'apprenant et le monde virtuel (historique des commandes passées) et un champ dans lequel l'apprenant peut entrer de nouvelles commandes. La solution de chaque problème est embarquée

dans le monde virtuel et les correspondances entre les phrases du scénario et le modèle Entité-Relation sont stockées dans la solution. L'apprenant est seulement autorisé à établir les correspondances idéales connues par le système. S'il tente de créer une association qui n'existe pas dans la solution connue par le système, celui-ci intervient et le prévient que la création de cette association n'est pas possible. Le système a donné lieu à une évaluation qualitative, par des concepteurs de bases de données novices et expérimentés : les concepteurs expérimentés se sentent restreints par la structure du monde virtuel alors que les novices ressentent une amélioration de leur compréhension de la modélisation de schémas Entité-Relation.

ERM-VLE restreint l'apprenant car celui-ci est forcé de suivre le modèle idéal connu par le système. De plus, le système interdit à l'apprenant de sortir du droit chemin mais sans expliquer pourquoi un autre cheminement serait une erreur. Par ailleurs, un environnement basé sur des commandes textuelles ne nous semble pas être un environnement naturel pour la construction de schémas Entité-Relation contrairement à la manipulation graphique des schémas. Cet aspect nous laisse supposer que les apprenants qui utilisent ERM-VLE pour apprendre à construire des schémas peuvent éprouver des difficultés à s'habituer à la modélisation de bases de données en dehors de l'environnement virtuel.

2.4.1.3 COLER

COLER [Constantino-González et Suthers, 2000] est un environnement d'apprentissage collaboratif basé sur le web pour la modélisation de schémas Entité-Relation. Les objectifs principaux du système sont d'améliorer les performances des apprenants dans la modélisation de schémas Entité-Relation et de les aider à développer des habiletés de pensée critique et collaborative. Le système contient un *coach* intelligent destiné à l'amélioration des capacités de modélisation des apprenants. COLER est conçu pour permettre l'interaction entre les apprenants localisés dans différents endroits via un environnement en réseau afin d'encourger la collaboration. L'interface de COLER contient un espace de travail privé dans lequel l'apprenant construit sa propre solution, et un espace de travail partagé où est construite la solution collaborative du groupe. Le système comporte une aide qui peut être utilisée pour obtenir des informations sur la modélisation. Les apprenants disposent d'un *chat* pour discuter avec les autres membres du groupe. Un seul membre du groupe à la fois est autorisé à modifier la solution commune, et une fois les modifications terminées, le droit d'édition est donné à un autre membre du groupe. L'interface contient également un volet affichant l'opinion du groupe sur le modèle courant. Pour chaque opinion donnée, les membres du groupe doivent indiquer s'ils sont d'accord, pas d'accord ou incertains. Le *coach*, affiché à gauche de l'écran, donne des conseils basés sur la dynamique du groupe : la participation des apprenants et du groupe à la construction du modèle.

COLER encourage et supervise la collaboration. Il peut aider les apprenants à acquérir des habiletés pour la collaboration. Toutefois, il n'évalue pas les schémas produits et ne fournit pas de rétroactions sur la correction de ces schémas. À cet égard, même si le système est efficace comme outil de collaboration, il ne serait pas efficace comme système d'enseignement pour un groupe de novices avec le même niveau d'expertise. De l'expérience des auteurs, il est très commun pour un groupe d'apprenants d'être d'accord sur un argument faux. En conséquence, il est très probable que des groupes d'apprenants sans la surveillance d'un expert apprennent des concepts erronés. Pour que COLER soit profitable aux apprenants qui l'utilisent, un expert doit être présent au cours

de la collaboration.

2.4.1.4 KERMIT

KERMIT (Knowledge-based Entity Relationship Modelling Intelligent Tutor) [Suraweera et Mitrovic, 2004] est un environnement informatique pour l'apprentissage de la modélisation de bases de données. Cette application est considérée comme un complément de cours où l'apprenant est supposé connaître les fondamentaux en bases de données. L'apprenant doit modéliser, via une interface adaptée, un schéma Entité-Relation satisfaisant le problème posé par le système. Pour la modélisation de chaque objet (entité, relation ou attribut) l'environnement demande à l'apprenant de choisir une expression dans le texte du problème. Celle-ci prend alors un style différent en fonction du type d'élément qu'elle représente (gras pour les entités et les relations, italique pour les attributs), comme le montre la figure 2.16 .

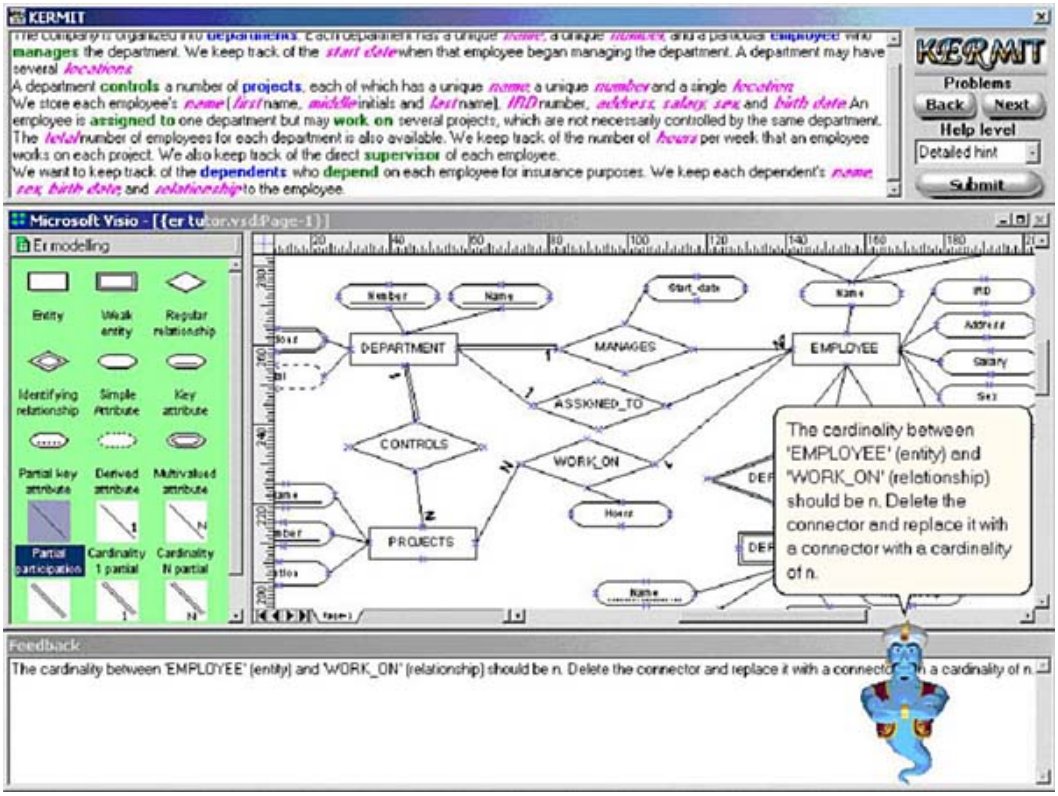


FIG. *2.16 – Interface de KERMIT (tirée de [Suraweera et Mitrovic, 2004])*

Le but de KERMIT est d'aider l'apprenant dans sa tâche de modélisation mais il offre peu d'aide au cours de la modélisation elle-même. Il propose pourtant à l'apprenant de souligner les mots du texte lors de la modélisation mais cet outil n'est pas conçu pour assister l'apprenant ou lui proposer une aide méthodologique : les liens sont utilisés par le système lors de l'évaluation de la solution de l'apprenant. En ce qui concerne les rétroactions fournies par le système, elles se font uniquement après l'évaluation de la solution et sont prédéfinies sous la forme de messages associés à des contraintes. L'apprenant est guidé par ces rétroactions vers la solution idéale connue du système mais ne dispose pas d'aide pour évaluer la pertinence de sa propre solution.

KERMIT a donné lieu à différents travaux de recherche au sein du groupe

ICTG [ICTG]. **EER-TUTOR**, une version WEB de KERMIT supportant le modèle Entité-Relation étendu (Enhanced Entity-Relationship model) a été développée [Zakharov et al., 2005]. **KERMIT-SE** est une version de KERMIT intégrant l'auto-explication [Weerasinghe et Mitrovic, 2002, Weerasinghe et Mitrovic, 2006].

2.4.2 EIAH pour la modélisation orientée objet et les diagrammes UML

Nous présentons dans ce paragraphe les deux approches majeures dans le contexte particulier de l'apprentissage de la modélisation orientée objet. La première, illustrée par l'environnement **DesignFirst-ITS**, est basée sur un module expert pour évaluer la solution de l'apprenant. La seconde est basée sur la notion de contraintes. Elle est utilisée dans **Collect-UML**. Chacune de ces approches nécessite une solution « idéale » utilisée comme référence pour évaluer les solutions de l'apprenant.

Nous présentons également les travaux de [Tholander et al., 1999] qui s'intéressent à l'aspect métacognitif de l'apprentissage de la modélisation orientée objet.

2.4.2.1 DesignFirst-ITS

DesignFirst-ITS est un tuteur intelligent qui fournit de l'aide aux apprenants lors de leur première approche de la conception de l'analyse orientée objet. Il s'intègre dans un enseignement basé sur le curriculum « Design First » [Moritz et al., 2005] préconisant l'enseignement de la conception d'une solution avec les concepts de la modélisation orientée objet avant son codage. DesignFirst-ITS interagit avec les apprenants *via* deux systèmes : le premier est un environnement de développement constitué de l'environnement Eclipse auquel deux *plugins* ont été ajoutés. Le *plugin* DrJava est un interpréteur interactif de code Java et LehighUML est un *plugin* pour la modélisation UML développé à l'université de Lehigh. L'apprenant doit construire des diagrammes de classes simples dans l'environnement de développement. DesignFirst-ITS observe la progression de l'apprenant, analyse ses actions et fournit des rétroactions. L'autre moyen d'interaction avec les apprenants se réalise par le biais de CIMEL (Collaborative constructive, Inquiry-based Multimedia E-Learning), un framework multimédia d'apprentissage constructif et collaboratif. CIMEL couvre un large éventail de thèmes de l'informatique, y compris l'introduction aux concepts de Java et de la programmation orientée objet. Le tuteur peut renvoyer les élèves à des éléments de CIMEL lorsque l'apprenant a besoin de renforcer ses connaissances. Il peut également utiliser les questionnaires et exercices interactifs disponibles dans CIMEL pour déterminer le niveau de compréhension de chaque concept de l'apprenant. La figure 2.17 montre l'interface de travail de CIMEL.

DesignFirst-ITS est constitué d'un modèle de curriculum, d'un module expert, du modèle de l'apprenant et d'un agent pédagogique.

Le modèle de curriculum contient les connaissances liées au domaine, c'est-à-dire les concepts de l'approche et de la programmation orientée objet. Ces concepts sont organisés en un réseau CIN (*Curriculum Information Network*) dans lequel ils sont liés par des relations de différentes natures indiquant par exemple les concepts « pré-requis » pour un concept donné, les composants d'un concept, etc. Un niveau de difficulté d'apprentissage est également assigné à chaque concept du réseau.

Le module expert est constitué d'un outil auteur « *Instructor Tool* » et d'un évaluateur expert « *Expert Evaluator* » [Moritz, 2008]. L'outil auteur permet à l'enseignant d'entrer la description d'un problème et de générer la solution qui sera utilisée par le module expert

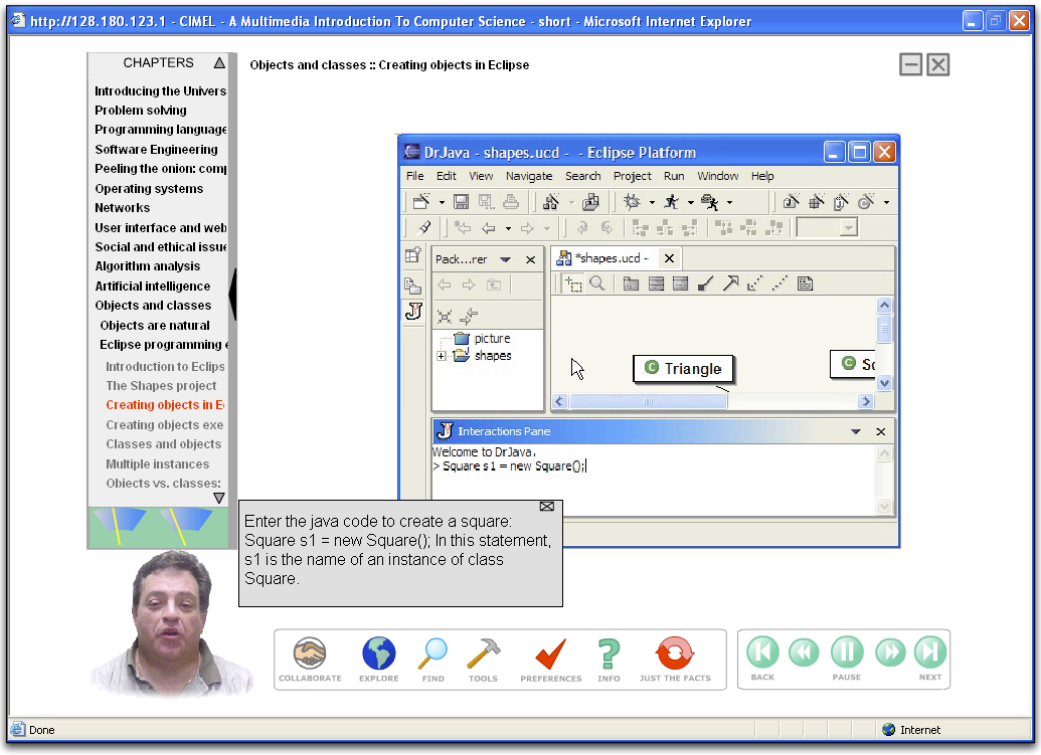


FIG. *2.17 – Framework CIMEL intégrant Eclipse et les plugins LehighUML et DrJava (tiré de [Moritz et Blank, 2005])*

pour évaluer la solution de l'apprenant. Pour générer la solution, l'« *Instructor Tool* » utilise des outils de traitement de la langue naturelle. La solution, qui contient les classes et les acteurs du problème, peut être modifiée et annotée par l'enseignant. Celui-ci peut par exemple ajouter des éléments optionnels qui pourront représenter des alternatives à la solution. L'évaluateur expert observe et analyse chaque action de l'apprenant et évalue si elle est correcte ou non en fonction de la solution connue du système. Quand une erreur est identifiée, elle est reliée à un concept du CIN et à une proposition de solution. Ce sont ces informations qui sont transmises au modèle de l'apprenant.

Le modèle de l'apprenant est constitué d'un modèle de domaine du problème « *Problem-Domain Model* (PDM) », d'un modèle de l'historique « *Historical Model* (HM) » et d'un modèle cognitif « *Cognitive Model* (CM) » [Wei, 2007]. Le PDM maintient l'état de connaissance de l'apprenant des concepts du réseau conceptuel CIN. Pour cela, il utilise un réseau bayesien atomique et affecte une probabilité à chaque concept. Le modèle de l'historique maintient un historique des connaissances de l'apprenant. Il trace par exemple le nombre de fois qu'une erreur est répétée. Le modèle cognitif utilise les informations du PDM et du HM pour effectuer un diagnostic cognitif et déterminer les causes de l'erreur de l'apprenant, identifier les stratégies de résolution de problème employées par l'apprenant et identifier les éventuelles lacunes dans ses connaissances. Le résultat du diagnostic cognitif est ensuite transmis au module pédagogique.

L'agent pédagogique fonctionne selon deux modes. Dans le mode « Conseil », il fournit une rétroaction immédiate aux actions erronées de l'apprenant. La rétroaction est centrée sur le concept du CIN auquel elle est liée. Trois niveaux de détail sont définis. Par exemple, si l'apprenant confond les attributs et les paramètres, le premier niveau de rétroaction peut

être le message « Rappelle-toi, les attributs sont utilisés pour représenter les caractéristiques des objets, alors que les paramètres sont utilisés pour transmettre des informations dans les méthodes. ». Si après le troisième indice l'apprenant fait toujours des erreurs mettant en jeu ce concept, l'agent pédagogique utilisera le mode « Tutoriel » pour expliquer de façon plus détaillée le concept qui n'a pas été appliqué correctement [Parvez, 2007].

DesignFirst-ITS est un système tutoriel à destination des débutants qui doivent, à partir d'une description textuelle complète, retrouver les classes et leurs paramètres. La solution du système est générée à partir de la description textuelle, ce qui contraint à la définir très précisément. Le système est basé sur un ensemble de concepts du domaine et l'apprenant est modélisé par son niveau de connaissance de chaque concept. Lorsqu'une action erronée est repérée, une rétroaction portant sur les concepts mis en jeu dans cette action, est proposée immédiatement. La rigidité de l'interaction dans ce système nous indique qu'il s'agit plutôt d'un exerciseur reposant sur une liste d'exercices très contraints. Si les différents modules qui le composent ont été testés par des experts, aucune expérimentation du système global avec des apprenants n'a encore été publiée.

2.4.2.2 Collect-UML

Collect-UML est un système d'apprentissage de la modélisation orientée objet avec UML qui utilise l'approche basée sur les contraintes (*Constraint-Based Modelling*) pour la modélisation du domaine et de l'apprenant [Baghaei, 2007]. Comme KERMIT, il est issu des travaux de recherche de l'ICTG (*Intelligent Computer Tutoring Group*) [ICTG].

Deux versions de Collect-UML ont été développées : une version individuelle [Baghaei et al., 2006] et une version collaborative [Baghaei et Mitrovic, 2006]. Dans la première version, c'est l'apprentissage individuel qui est visé. L'apprenant doit construire un diagramme de classes à partir d'une description textuelle du problème. Comme dans KERMIT, l'apprenant doit utiliser les mots de l'énoncé pour construire le diagramme. Les liens entre l'énoncé et le diagramme seront utilisés par le système pour comparer la solution de l'apprenant et la solution idéale modélisée par un ensemble de contraintes syntaxiques et sémantiques. Chaque contrainte est composée d'une condition de pertinence, d'une condition de satisfaction et un message de rétroaction. Lorsque la condition de pertinence est vraie, la condition de satisfaction doit l'être également. Dans le cas contraire, la contrainte est violée, et le message de rétroaction associé est affiché à l'apprenant. La figure 2.18 montre deux exemples de contraintes dans Collect-UML.

La contrainte 41 est une contrainte syntaxique qui vérifie que des attributs ou des opérations sont définis pour chaque classe de la solution de l'apprenant. La contrainte 49 est une contrainte sémantique. La condition de pertinence identifie une sous-classe dans la solution idéale du système et vérifie si la solution de l'apprenant contient cette même classe. La solution de l'apprenant est correcte si la condition de satisfaction est remplie, c'est-à-dire si la classe en question est une sous-classe d'une autre classe du diagramme.

```
(41
SI [condition de pertinence]
La solution de l'apprenant contient une classe C
ALORS [satisfaction condition]
Il est nécessaire que la solution de l'apprenant contienne un
attribut ou une opération dans C
SINON
Message : "Vérifies tes classes. Chaque classe doit avoir au
moins un attribut ou une opération."

(49
SI [condition de pertinence]
La solution idéale contient une sous-classe C,
ET que la solution de l'apprenant contient une classe du même nom
ALORS [condition de satisfaction]
Il est nécessaire que C soit également une sous-classe dans la
solution de l'apprenant
SINON
Message : "Vérifies si tu as défini toutes les sous-classes
requises. Il manque certaines sous-classes."
```

FIG. *2.18 – Exemple de contraintes dans Collect-UML (adapté de [Baghaei, 2007])*

L'interface de Collect-UML possède plusieurs volets : en haut de l'écran se trouvent les boutons permettant de sélectionner un problème, visualiser l'historique de la session ou le modèle de l'apprenant, demander de l'aide ou imprimer le diagramme. Le panneau central est constitué de l'affichage du texte du problème et de l'espace de modélisation. Les rétroactions sont présentées dans le panneau de droite. En bas de l'interface on trouve les boutons de sélection du niveau de détail des rétroactions et le bouton de soumission de la solution (cf. figure 2.19).

Les rétroactions sont classées selon cinq niveaux de détail :

- *rétroaction simple* : indique simplement si la solution soumise est correcte ou non ;
- *indication d'erreur* : indique le type d'élément concerné par l'erreur associée à la première contrainte violée ;
- *conseil* : affiche le message de rétroaction associé à la première contrainte violée ;
- *tous conseils* : affiche l'ensemble des messages de rétroaction associés aux contraintes violées ;
- *solution* : affiche la solution dans une fenêtre séparée.

Au départ, le niveau de détail des rétroactions est fixé à *Rétroaction simple* et augmente à chaque soumission de la solution jusqu'au niveau *Conseil*. L'apprenant peut à tout moment modifier le niveau de détail des rétroactions. Les contraintes sont classées dans la base de connaissances par l'enseignant et cela détermine l'ordre dans lequel les rétroactions seront données : dans le cas où plusieurs contraintes sont violées et que le niveau de détail des rétroactions est inférieur à *Tous conseils*, le système ne traite que la première contrainte violée.

Dans la version multi-utilisateurs, les apprenants peuvent travailler dans un espace privé pour construire leur propre solution, ou travailler en collaboration pour produire une solution collective. Le système donne alors des conseils au groupe pour favoriser les échanges et la discussion entre les apprenants. Pour cela, les interactions des apprenants sont comparées avec un modèle idéal d'interaction à base de contraintes également.

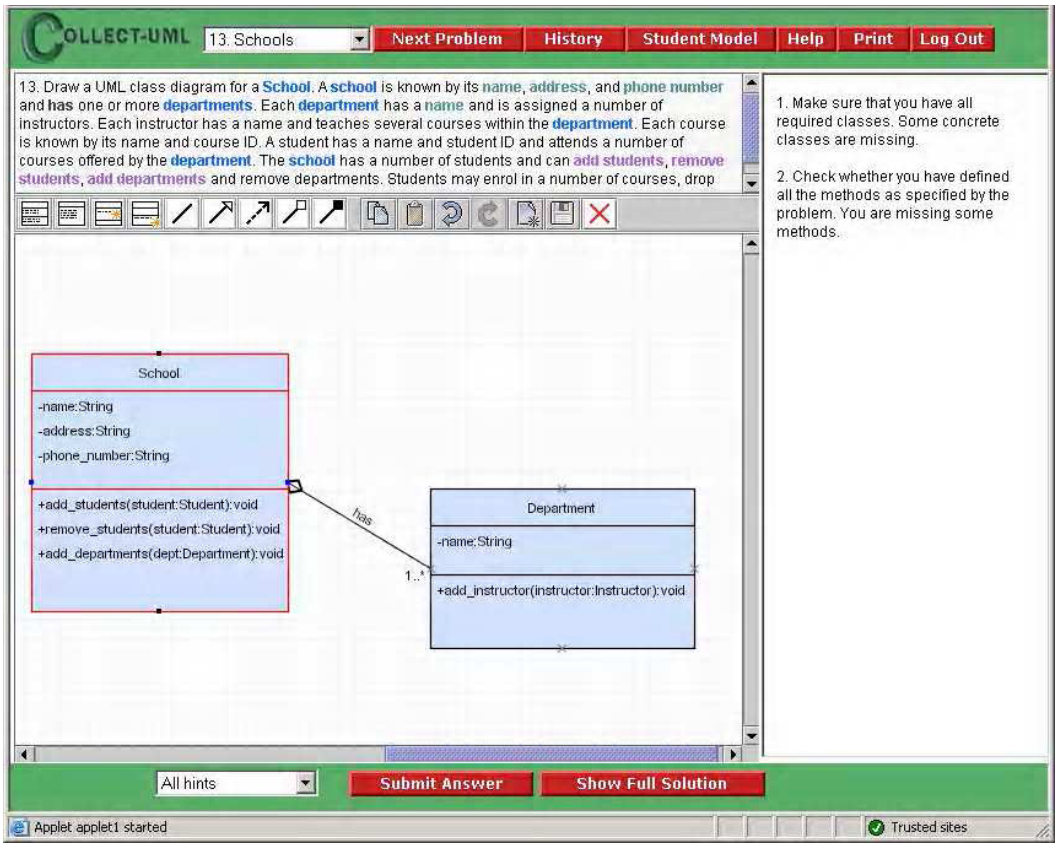


FIG. *2.19 – Interface de Collect-UML (tirée de [Baghaei, 2007])*

La généricité de l'approche par contraintes de systèmes tels que KERMIT ou Collect-UML est bien adaptée aux domaines ouverts. Cependant, l'efficacité du diagnostic dépend fortement de la définition des contraintes ; or, il existe peu de contraintes sémantiques générales. De plus, elles sont délicates à exprimer. Les rétroactions dans ces deux systèmes sont les messages associées aux contraintes violées. L'apprenant ne dispose pas d'aide pour évaluer la pertinence de sa propre solution, il est guidé vers la solution idéale du système.

2.4.2.3 [Tholander et al., 1999]

Les auteurs de [Tholander et al., 1999] se sont intéressés à l'aspect métacognitif de la modélisation orientée objet. Ils proposent d'utiliser le compagnonnage cognitif (*cognitive apprenticeship*) comme support à l'apprentissage de la modélisation orientée objet en intégrant trois outils d'apprentissage à un outil de modélisation classique. Ces trois outils sont un assistant pédagogique, une bibliothèque de patrons de modélisation et des traces de résolution par des experts de problèmes similaires à ceux que rencontrent les apprenants. L'apprenant peut visualiser et suivre les commentaires étape par étape de la résolution d'un exercice par un expert. Cela permet à l'apprenant de se confronter avec les pratiques et le vocabulaire utilisés par des experts. L'assistant pédagogique est chargé d'encourager la réflexion métacognitive. Pendant que l'apprenant construit son modèle, l'assistant lui pose des questions et émet des remarques critiques sur le modèle (cf. figure 2.20) : cela exerce une fonction de surveillance de son activité par l'apprenant en l'encourageant à avoir un regard critique par rapport à sa solution. Trois types de questions ou commentaires sont

possibles :

- Des questions spécifiques aux éléments du modèle constuit par l'apprenant. Par exemple : « Es-tu conscient que tu as une relation plusieurs-à-plusieurs entre les classes 'Avion' et 'Vol' ? Es-tu sûr que c'est bien ce que tu veux ? ». Ces questions sont basées sur des règles générales et des heuristiques de modélisation appliquées au modèle en cours de construction.
- Des questions générales sur la manière dont l'apprenant va traiter certains aspects du problème. Par exemple : « Peux-tu, avec le modèle que tu as construit, représenter le fait que les vols peuvent être redirigés vers une autre destination que celle prévue initialement ? ». Ces questions sont préconstruites à partir des sujets identifiés dans le domaine.
- Des commentaires généraux, basés sur des stratégies générales de résolution de problème, pour encourager la réflexion et le regard critique de l'apprenant sur son travail. Par exemple : « Es-tu sûr que c'est une bonne idée ? » lors de la construction d'un nouvel élément ou « Si tu es perdu, essaye de revenir à ton plan initial » si l'apprenant n'a pas modifié son modèle depuis un certain temps.

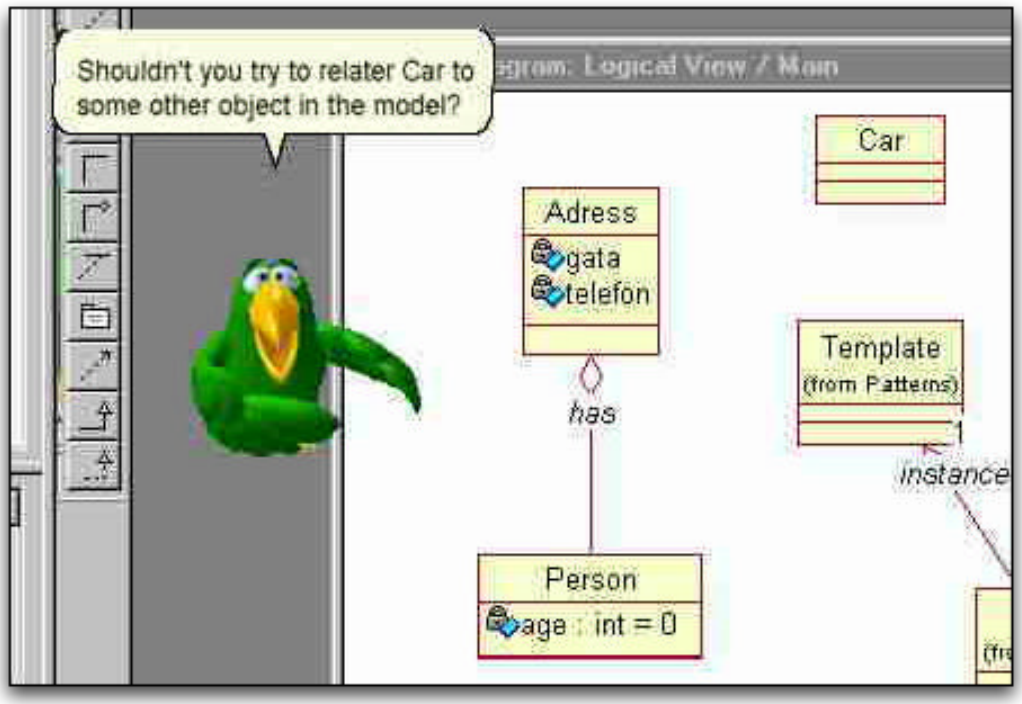


FIG. *2.20 – Assistant pédagogique dans [Tholander et al., 1999]*

Les trois outils ont été intégrés dans l'outil commercial *Rational Rose* et ont donné lieu à une évaluation avec des apprenants travaillant en binôme [Tholander et al., 1999]. Les discussions entre les apprenants au cours de l'évaluation montrent que les apprenants ont pris en compte les commentaires et les questions de l'assistant même si aucune influence directe sur le modèle n'a été observée. Les évaluations ont montré l'importance cruciale du moment d'intervention de l'assistant. Certains commentaires sont affichés trop tôt, et les apprenants commentent « nous n'avons pas encore terminé cela » ou « oui, mais nous sommes en train de le faire ». Certains apprenants ont déclaré qu'ils préféreraient travailler seul pendant un moment avant de demander l'intervention du tuteur.

2.4.2.4 Synthèse

DesignFirst-ITS et Collect-UML sont deux systèmes qui utilisent une évaluation du diagramme de l'apprenant pour produire des rétroactions. DesignFirst-ITS utilise un modèle expert. Les rétroactions sont immédiates, et sont plutôt axées sur les concepts mis en jeu. Collect-UML utilise une modélisation du domaine par des contraintes. Les messages de rétroaction sont transmis lorsqu'une contrainte est violée. Dans les deux systèmes, l'apprenant est guidé vers la solution idéale et des rétroactions sont fournies dès qu'une action est erronée dans un cas, ou dès qu'une contrainte est violée dans l'autre. Contrairement à ces sytèmes, l'assistant pédagogique de [Tholander et al., 1999] propose des aides qui ont pour objectif d'aider l'apprenant à évaluer sa propre solution.

2.5 Positionnement de nos travaux

Dans le cadre de nos travaux, nous retenons de ce chapitre les éléments suivants :

- La modélisation orientée objet est une activité complexe d'une importance majeure dans le cycle de développement logiciel. L'apprentissage des principes de la modélisation orientée objet est donc essentiel dans des cursus informatiques. Nous nous intéresserons spécifiquement à l'apprentissage par des novices de la modélisation par diagrammes de classes en amont de la programmation c'est-à-dire la construction du diagramme de classes en phase d'analyse du génie logiciel.
- Les apprenants novices en modélisation orientée objet se heurtent à de nombreuses difficultés au cours de leur apprentissage. La construction d'un modèle n'est pas une tâche fermée : il n'existe pas de méthode unique pour construire un modèle et plusieurs modèles sont souvent possibles dans une même situation. Aussi la théorie ne suffit pas, la pratique de la modélisation est une part importante de l'apprentissage. Il est donc essentiel que les apprenants disposent d'un environnement informatique de modélisation adapté à leur situation d'apprentissage.
- Les environnements de modélisation classiques sont principalement à destination des professionnels. Les outils à visée éducative proposent souvent une interface simplifiée mais offrent peu d'aide pour l'évaluation de la solution de l'apprenant. Il existe peu de travaux en EIAH concernant la modélisation orientée objet. Les deux principales approches existantes sont l'utilisation d'un module expert et la modélisation basée sur les contraintes. Dans les deux cas, l'apprenant est plutôt guidé vers la solution idéale du système.
- Nous pensons qu'un environnement informatique pour l'apprentissage de la modélisation doit permettre aux apprenants d'acquérir une méthodologie et des habiletés permettant d'évaluer leurs propres productions au cours de l'apprentissage. Pour cela, les modes d'interaction doivent être adaptés aux novices et encourager la réflexion et le regard critique de l'apprenant sur son travail.
- Malgré le fait que nous nous focalisions sur le domaine de la modélisation orientée objet, nous pouvons dégager un certain niveau de généricité dans notre étude : nos propositions ont pour objectif d'être adaptables à différents domaines de la modélisation en informatique. En effet, il existe de nombreux points communs entre la construction d'un diagramme de classes et la construction d'un schéma relationnel de bases de données et nous constatons les mêmes difficultés au cours de l'apprentissage.

Le chapitre suivant est consacré à l'aspect métacognitif de l'activité de modélisation. En effet, nous avons constaté dans ce chapitre l'importance pour l'apprenant d'acquérir des compétences de niveau métacognitif afin de pouvoir évaluer son diagramme au cours de la construction.

CHAPITRE 3

Rôle de la métacognition dans les EIAH

Sommaire

Résumé

Ce chapitre est consacré à la métacognition et à l'aspect métacognitif de l'activité de modélisation. En effet, nous avons constaté dans le chapitre précédent qu'il est important que l'apprenant évalue la pertinence de sa propre solution au cours de l'activité de modélisation. Nous pensons que l'interaction dans un EIAH pour la modélisation doit favoriser l'acquisition par l'apprenant de procédures de contrôle et de validation de ses productions, qui relèvent du niveau métacognitif. Nous nous intéressons donc au concept de métacognition. Le premier paragraphe de ce chapitre est consacré à l'étude des processus métacognitifs dans la littérature. Dans un deuxième temps nous abordons les aspects cognitifs et métacognitifs de la modélisation, puis nous étudions les différentes approches existantes de la métacognition dans les EIAH. Enfin, nous terminons ce chapitre par le positionnement de nos travaux par rapport à l'ensemble du chapitre.

3.1 Processus métacognitifs

3.1.1 Définitions de la métacognition

Littéralement « connaissance sur la connaissance », le terme de métacognition est employé pour désigner la connaissance qu'un sujet possède de ses propres processus de

pensée et de ceux d'autrui, ainsi que le contrôle qu'il exerce sur ses propres processus cognitifs [Allal et Saada-Robert, 1992]. Le terme de métacognition a été évoqué à l'origine dans les écrits de [Flavell, 1976] :

" La « métacognition » se réfère à la connaissance du sujet sur ses propres processus cognitifs, de leurs produits et de tout ce qui s'y rapporte, etc. La métacognition concerne le contrôle (monitoring) et la résultante régulation ou orchestration de ces processus en fonction des objets cognitifs ou des données sur lesquelles ils portent, habituellement pour servir un but ou un projet concret "

[Flavell, 1976], traduit et cité par [Allal et Saada-Robert, 1992]

Selon cette définition, le champ de la métacognition comprend deux dimensions essentielles : les connaissances métacognitives et les régulations métacognitives.

La première dimension a été analysée par la suite dans plusieurs articles de Flavell : il précise alors la distinction entre les « connaissances métacognitives » relatives aux personnes, aux tâches et aux stratégies, qui sont des représentations en mémoire à long terme, et les « expériences métacognitives » qui reflètent la prise de conscience plus ou moins élaborée des processus cognitifs en action ([Flavell, 1981] cité par [Allal et Saada-Robert, 1992]).

La seconde composante de la métacognition est développée dans les travaux de [Brown, 1978] et désigne les mécanismes de régulation ou de contrôle du fonctionnement cognitif. Nous détaillons cette composante dans le paragraphe suivant.

3.1.2 Mécanismes de régulation du fonctionnement cognitif

Les mécanismes de régulation du fonctionnement cognitif font référence au contrôle de ce système cognitif, en d'autres termes aux outils permettant de planifier, de réguler (processus d'autorégulation) et d'évaluer. On trouve dans la littérature différents découpages des mécanismes de régulation. Pour [Brown, 1978], la régulation métacognitive comprend trois fonctions principales : la **planification** (*planning*) d'activités à entreprendre, le **contrôle** (*monitoring*) d'activités en cours de réalisation, et la **vérification** (*checking*) des résultats d'activités. [Allal et Saada-Robert, 1992] parlent d'« opérations de régulation ». Les opérations de régulation sont :

- **l'anticipation** : cette opération traduit l'organisation des représentations du sujet en orientations de l'action et assure ainsi le guidage de proche en proche des processus de production ;
- **le contrôle** : cette opération implique un processus continu de comparaison entre un état donné et un état-but à atteindre ;
- **l'ajustement** : cette opération est la conséquence de l'opération de contrôle. Si le monitoring met en évidence une divergence entre l'état présent et l'état-but, l'opération d'ajustement introduit une modification ou une réorientation des processus de production.

Pour ces auteurs, ces opérations de régulations sont intégrées au fonctionnement cognitif et ne peuvent être considérées comme métacognitives que si elles sont actives et conscientes. D'autres chercheurs pensent en revanche que le processus d'autorégulation n'est ni conscient, ni explicite. Pour [Brown, 1978] ces fonctions se déroulent le plus souvent de

manière implicite et automatisée. Toutefois, lorsque l'apprenant rencontre des difficultés ou est confronté à une situation nouvelle, une prise en charge plus consciente et plus délibérée des processus de régulation aura tendance à s'enclencher. D'après [Brown, 1978] la capacité à savoir exercer une régulation explicite est au cœur du développement métacognitif.

3.1.3 Auto-réflexion et autorégulation

Le processus d'auto-réflexion fait référence à la réflexion d'un sujet sur lui-même : c'est une partie importante de la métacognition. L'auto-réflexion permet une meilleure compréhension de ses connaissances par l'apprenant, mais c'est également un moyen d'améliorer ses stratégies métacognitives. Par exemple, lorsque l'apprenant réfléchit sur une action qu'il vient d'exécuter, il revisite consciemment les informations liées à cette action. Lors de la résolution d'un problème par étapes, l'auto-réflexion peut se manifester par le passage en revue des précédentes actions et décisions avant de passer à l'étape suivante. Ainsi l'auto-réflexion dans un environnement d'apprentissage peut accroître l'effet bénéfique en terme d'apprentissage des exercices.

D'après [Zimmerman, 2001], l'auto-réflexion joue un rôle central pour l'autorégulation dans l'apprentissage. Selon lui, ce qui caractérise l'autorégulation des apprenants, c'est leur participation active dans l'apprentissage du point de vue métacognitif, motivationnel et comportemental. Pour l'auteur, les apprenants autorégulés sont engagés dans trois phrases cycliques :

– prévision (*forethought*) : processus engagés avant l'action ;
– contrôle de l'exécution (*performance control*) : processus engagés au cours de l'apprentissage ;
– auto-réflexion (*self-reflection*) : processus engagés après l'action ou l'apprentissage.

Le processus de prévision influence le processus de contrôle qui, à son tour, influence l'auto-réflexion. Un cycle est complet lorsque la phase d'auto-réflexion a un impact sur la phase de prévision des apprentissages à venir.

3.2 Métacognition et apprentissage

3.2.1 Intérêt de la métacognition dans l'apprentissage

Dès 1979, Flavell revendique le fait que la métacognition joue un rôle important dans de nombreux domaines :

" Investigators have recently concluded that metacognition plays an important role in oral communication of information, oral persuasion, oral comprehension, reading comprehension, writing, language acquisition, attention, memory, problem solving, social cognition, and various types of self control and self-instruction "

[Flavell, 1979], p.906

Il ajoute également :

“ I think that increasing the quantity and quality of children’s metacognitive knowledge and monitoring skills through systematic training may be feasible as well as desirable ”

[Flavell, 1979], p.910

La métacognition est un facteur facilitant les apprentissages et contribuant au développement de l’apprenant par une meilleure connaissance de soi et de ses possibilités.

Dans l’approche socio-constructiviste, l’apprenant est acteur de ses propres apprentissages. Il construit son savoir en interaction avec le milieu et peut ainsi se réguler. Il est important d’encourager les capacités de métacognition et de réflexivité qui permettent cette régulation. Donner l’habitude à l’apprenant d’une réflexion sur son travail le rend plus autonome dans la gestion de ses apprentissages. Les activités de régulation renforcent également le contrôle de ses choix par l’apprenant et lui permettent d’avoir un regard critique sur son travail.

Selon [Palacio-Quintin, 1990], citée par [Romero, 2004], le développement de la métacognition consiste à permettre à l’élève de « *réfléchir sur sa propre action et de prendre conscience du cheminement l’ayant conduit aux résultats obtenus [...] et [...] de constater lui-même ses erreurs, de découvrir les raisonnements qui l’ont conduit à ces erreurs et de chercher les solutions adéquates* ». Dans cette perspective, l’erreur fait partie de l’apprentissage et peut être utilisée pour favoriser la métacognition. Pour cela, l’apprenant doit prendre conscience de son erreur et l’utiliser comme un objet d’analyse et de questionnement.

3.2.2 Techniques et outils métacognitifs pour l’apprentissage

Dans ce paragraphe, nous présentons plusieurs techniques pédagogiques et outils généralement utilisés par les enseignants pour favoriser la métacognition au cours de l’apprentissage.

3.2.2.1 Techniques métacognitives

Questions et incitations réflexives

Les questions et incitations réflexives sont des méthodes simples, pouvant être utilisées par les enseignants afin d’encourager la discussion qui commence par le bilan de l’activité d’apprentissage et peut aller jusqu’à une réflexion critique sur les stratégies utilisées pour accomplir la tâche, et les raisons de l’utilisation de ces stratégies.

Les questions réflexives sont plus générales que les incitations. Elles sont utilisées par l’enseignant pour déclencher un contrôle métacognitif global. Les questions sont par exemple « Et maintenant ? » ou « Et alors ? ». Elles peuvent aider l’apprenant à réfléchir sur la prochaine étape d’un problème et faire la liaison avec l’étape précédente [Amado Gama, 2004].

Les incitations sont des questions plus précises qui fournissent une aide plus directive sur certains aspects du processus d’apprentissage. Les incitations sont des questions telles que « Pourquoi... est-il important ? » ou « Peux-tu donner un exemple de ... ? ». Les questions ouvertes sont plus efficaces et engendrent un processus de réflexion plus important que les questions fermées. Les incitations peuvent également stimuler

l'auto-explication en guidant l'apprenant au cours de la construction d'une explication ou d'une justification.

Une des difficultés des questions et incitations réflexives est de trouver le moment adéquat pour intervenir auprès de l'apprenant.

Echafaudage métacognitif

L'échafaudage est une structure qui aide l'apprenant à combler la distance entre ce qu'il sait réaliser seul et ce qu'il peut réaliser avec une aide extérieure [Hartman, 2001]. L'échafaudage doit être régulé en fonction des besoins de l'apprenant dans l'objectif de le rendre plus indépendant, d'améliorer l'autorégulation. L'échafaudage métacognitif peut prendre diverses formes : incitations (par exemple à utiliser une ressource ou un outil disponible pouvant l'aider à résoudre la tâche), indices, allusions, solution partielle, rappel des objectifs de la tâche, etc.

Auto-questionnement

Susciter l'auto-questionnement est un moyen efficace d'améliorer les capacités d'auto-direction de l'apprenant. Des questions telles que « N'ai-je rien oublié d'important ? » peuvent par exemple l'aider à identifier l'absence de certains points importants. L'auto-questionnement peut guider l'apprenant avant, pendant et après l'exécution de la tâche. Il permet d'améliorer la conservation à long terme de ses connaissances et compétences et la capacité à les réutiliser mais aussi d'améliorer l'attitude et d'accroître la motivation de l'apprenant [Hartman, 2001].

Auto-explication

L'auto-explication (*self-explanation*) est un processus qui consiste à expliquer à soi-même le contenu d'un exercice, d'un texte ou d'un exemple afin de le rendre plus clair et intelligible [Chi et al., 1989]. Les auto-explications ont plus d'effets que les explications données par des tiers car elles nécessitent une élaboration active de la part de l'apprenant. Des études montrent que la plupart des apprenants n'utilisent pas spontanément l'auto-explication et ont souvent besoin qu'on les guide ou qu'on les incite à pratiquer cette activité [Chi et al., 1989].

Auto-évaluation

L'auto-évaluation permet aux apprenants d'évaluer l'état de leurs connaissances, d'expliciter ce qu'ils savent ou ne savent pas. Les apprenants qui observent et évaluent précisément leurs performances peuvent réagir de manière appropriée en conservant ou en changeant leur stratégie pour atteindre leur but.

3.2.2.2 Outils métacognitifs

Modélisation

Le modèle en tant qu'objet de raisonnement peut être utilisé comme support à une activité cognitive comme l'acquisition et la construction des connaissances de l'apprenant. L'activité de modélisation participe à la prise de conscience par l'apprenant de ses idées et modes de raisonnement [Komis et al., 2003].

Réification graphique

Les représentations graphiques peuvent être utilisées par l'apprenant pour organiser ses réflexions, structurer ses idées. Elles peuvent prendre la forme de cartes conceptuelles, d'organigrammes, de représentations arborescentes, de réseaux, etc.

3.2.3 Apports de la métacognition lors de la tâche de modélisation

Nous avons montré dans le chapitre précédent que la modélisation n'est pas un processus fermé, et qu'il existe souvent plusieurs représentations possibles pour un problème donné. Il semble donc important de favoriser les capacités métacognitives de l'apprenant telles que la régulation de son activité, de son apprentissage, la planification, l'évaluation des résultats de son activité. Pour cela, il est possible d'utiliser différentes techniques telles que celles que nous avons présentées dans le paragraphe 3.2.2. Par exemple, l'assistant pédagogique dans [Tholander et al., 1999] présenté dans le chapitre 2, paragraphe 2.4.2.3, page 40, utilise des questions et commentaires qui peuvent s'apparenter à des éléments d'échafaudage métacognitif qui ont pour objectif de favoriser la réflexion et la régulation et d'encourager l'apprenant à avoir un regard critique sur son travail.

3.3 Approches de la métacognition dans les EIAH

Grâce aux possibilités d'interaction et aux capacités d'enregistrement de l'activité de l'apprenant offertes par les ordinateurs, ceux-ci sont de puissants outils pour supporter la métacognition. Le système peut aider l'apprenant au cours de la tâche, poser des questions sur ses choix ou sur sa façon de résoudre le problème. Les environnements d'apprentissage collaboratif permettent la planification du travail, le contrôle et l'évaluation du processus d'apprentissage. Les apprenants travaillant en petits groupes peuvent par exemple travailler sur une solution individuelle, puis comparer et discuter de leurs différentes solutions, ou travailler à une solution commune.

3.3.1 Intégration de la métacognition dans les EIAH

Nous présentons ici deux études sur l'intégration de la métacognition dans les environnements. [Amado Gama, 2004] propose un schéma de classification pour la conception de la métacognition dans les environnements informatiques pour l'apprentissage en trois axes : le temps, le niveau métacognitif visé et l'approche pédagogique. Margarida Romero a étudié la prise en compte de la métacognition dans les EIAH au cours de son Master, réalisé à l'université du Maine [Romero, 2004, Romero, 2005].

3.3.1.1 Classification de la métacognition dans les EIAH de [Amado Gama, 2004]

[Amado Gama, 2004] propose un schéma de classification pour la conception de la métacognition dans les environnements informatiques pour l'apprentissage, que nous présentons dans le tableau 3.1.

L'auteur propose trois dimensions de classification : le temps, le niveau métacognitif visé et l'approche pédagogique (*instructional approach*). Ces dimensions sont indépendantes et les choix effectués concernant une dimension n'affectent pas les choix possibles dans les autres dimensions.

TAB. *3.1 – Schéma de classification pour la conception de la métacognition dans les environnements informatiques pour l'apprentissage (tiré de [Amado Gama, 2004])*

Dimensions	**Valeurs possibles**
Temps (*Quand ?*)	– Avant la tâche d'apprentissage – Pendant la tâche d'apprentissage – Après la tâche d'apprentissage
Niveau visé (*Quoi ?*)	– Niveau métacognitif général (indépendant du domaine) – Niveau lié au domaine – Niveau spécifique à la tâche
Approche pédagogique (*Comment ?*)	Techniques et outils tels que : – Collaboration : entre pairs, par le compagnonnage, etc. – Suivi réflexif – Auto-réflexivité, auto-explication, auto-évaluation, etc. – Echafaudage métacognitif – Réification graphique

La dimension de temps fait référence au moment choisi pour les activités métacognitives et les activités réflexives. Les valeurs possibles ne sont pas exclusives. D'après [Flavell, 1979] les expériences métacognitives peuvent se présenter avant, pendant ou après les activités cognitives :

- **Avant l'activité** : afin de préparer l'apprenant avant le commencement d'une nouvelle tâche ;
- **Pendant l'activité** : agir au cours de l'activité permet de supporter l'auto-contrôle mais cela peut créer une surcharge cognitive ;
- **Après l'activité** : c'est un moment approprié pour que l'apprenant réfléchisse sur ses performances et sur ses processus d'apprentissage.

La deuxième dimension porte sur le niveau de dépendance au contexte (la tâche ou le domaine d'apprentissage) des compétences métacognitives visées par le système :

- **Niveau général** : lorsque les compétences métacognitives visées ne sont pas spécifiques au domaine de l'activité ;
- **Relatif au domaine** : lorsque les compétences métacognitives, bien qu'applicables à d'autres domaines, sont celles qui sont les plus utiles dans le domaine visé : par exemple la planification des buts de la lecture dans le cas de l'apprentissage à partir de textes ;
- **Spécifique à la tâche** : lorsque les éléments métacognitifs se focalisent sur la tâche que l'apprenant doit réaliser et font réfléchir l'apprenant sur les étapes à suivre pour accomplir cette tâche.

Lorsque les compétences métacognitives visées sont dépendantes du domaine ou spécifiques à la tâche, le travail métacognitif est réalisé en plus de la tâche d'apprentissage :

soit on attire l'attention de l'apprenant sur l'accomplissement d'un travail supplémentaire de type métacognitif, soit le travail métacognitif est mis en place mais rien n'est fait pour attirer l'attention de l'apprenant sur cela. Dans le premier cas, le travail sur la métacognition est dit « explicite » alors que dans le second cas, il est dit « implicite ». L'explicitation apporte une charge de travail supplémentaire à l'apprenant mais l'effet est de plus longue durée car l'apprenant est conscient du travail métacognitif qu'il effectue.

L'approche pédagogique fait référence aux mécanismes utilisés pour implémenter les activités métacognitives comme la collaboration, la réification graphique, le rejeu de la tâche ou les activités auto-réflexives. Certains environnements peuvent combiner plusieurs approches : ils peuvent utiliser à la fois des tâches collaboratives et des activités de questionnement.

3.3.1.2 Prise en compte du développement métacognitif dans les EIAH par [Romero, 2005]

Margarida Romero propose dix principes pour la prise en compte du développement métacognitif dans les EIAH [Romero, 2005] :

1. Le **reflet** permet à l'apprenant de prendre conscience de soi comme apprenant. Cela peut être un jugement humain (*feedback* des responsables pédagogiques, de ses pairs), un retour automatique grâce aux outils d'auto-évaluation ou un *feedback* sur l'avancement grâce à des outils de suivi.
2. Le **travail collaboratif** facilite la transmission de connaissances métacognitives. Pour intégrer une dimension collaborative, on peut utiliser la pédagogie de projet, les outils de communication, d'organisation du travail collectif ou des outils de travail collaboratif.
3. Le **métatuteur** est défini comme la personne qui accompagne le développement métacognitif des apprenants dans l'EIAH. Il peut effectuer différents types d'intervention visant le développement métacognitif de l'apprenant comme la régulation collective synchrone, le suivi individuel ou la reformulation de l'objectif de l'activité.
4. Le **tutorat entre pairs** visant le développement métacognitif peut être intégré de différentes manières : former des binômes expert/novice pour permettre à l'apprenant expert de prendre conscience et d'expliciter ses stratégies et au novice de prendre en compte ce retour d'expérience, répartir les rôles dans un groupe de travail, mener des évaluations entre pairs.

L'objectif de développement métacognitif s'articule la plupart du temps avec celui d'apprentissage et d'acquisition de connaissances de l'EIAH. Ses deux objectifs peuvent s'articuler de trois manières :

5. **Articulation (1)** : le développement métacognitif est isolé, avant ou après l'activité principale.
6. **Articulation (2)** : les deux objectifs se développent en parallèle, mais sont distincts l'un de l'autre ;
7. **Articulation (3)** : les deux objectifs se fondent dans une activité unique.
8. L'utilisation d'**outils de communication** pour faciliter les échanges, animer les communications entre apprenants ;

9. Les **outils de modélisation** : l'intérêt métacognitif des outils de modélisation réside dans la réflexion portée sur le modèle élaboré ainsi que dans la discussion du modèle avec d'autres membres de la communauté d'apprentissage qui implique une forte négociation et une large modification de la perception de la réalité.
10. L'intégration dans l'EIAH d'**outils de développement métacognitif** : cela peut être des outils standards dont l'utilisation sera à visée métacognitive.

Nous retrouvons dans les principes proposés par [Romero, 2005], quatre éléments essentiels à introduire dans l'EIAH : de la **réflexivité**, des **interactions humaines** où l'on échange autour des métaconnaissances et des stratégies d'apprentissage, l'**articulation entre développement métacognitif et activité principale**, et des **outils permettant la gestion de son propre apprentissage**. Les deux premiers éléments ont une nature très générale. [Romero, 2005] semble se placer à un niveau métacognitif indépendant du domaine de la tâche d'apprentissage. L'articulation entre développement métacognitif et activité principale peut être mise en parallèle avec la dimension temporelle de la taxonomie de [Amado Gama, 2004]. Enfin, les outils pour la gestion de son propre apprentissage sont à mettre en relation avec l'approche pédagogique proposée par [Amado Gama, 2004]. La classification de [Amado Gama, 2004] nous semble plus concise et générique, aussi nous l'utiliserons comme référence concernant l'intégration de la métacognition dans les EIAH, dans la suite de ce chapitre.

3.3.2 Exemples d'intégration de la métacognition dans les EIAH

Nous présentons dans ce paragraphe différents travaux portant sur l'intégration d'une dimension métacognitive dans un environnement informatique pour l'apprentissage, selon différentes approches. Nous décrivons chaque système en fonction des différents critères de la taxonomie de [Amado Gama, 2004] présentée dans le paragraphe 3.3.1.1 : approche pédagogique utilisée, niveau et compétences métacognitives visées, activités, outils ou fonctionnalités mis en œuvre, etc.

Auto-explication

[Conati et VanLehn, 2000] ont développé le système **SE-Coach** pour l'apprentissage à partir d'exemples *via* l'auto-explication. Il est implémenté dans ANDES, un tuteur pour l'apprentissage de la physique newtonnienne [VanLehn, 1996]. Il existe trois niveaux d'échafaudage de l'auto-explication dans SE-Coach. Le premier niveau consiste en un mécanisme masquant différentes parties de l'exemple et qui force l'apprenant à passer sa souris sur la partie qu'il souhaite lire. Au deuxième niveau, le système incite l'apprenant à l'auto-explication. A chaque fois que celui-ci dévoile une partie de l'exemple pouvant donner lieu à des explications, un bouton *auto-expliquer* est affiché à l'interface. Le troisième niveau consiste en un ensemble d'outils permettant la construction d'auto-explications. Par exemple, le navigateur de règles contient toutes les règles de la physique du système. Lorsque l'apprenant sélectionne une règle qui d'après lui justifie l'élément découvert de l'exemple, SE-Coach fournit une rétroaction grâce à un code couleur vert/rouge indiquant si la règle sélectionnée est celle qui justifie l'élément découvert (cf. figure 3.1). Pour élaborer une auto-explication l'apprenant doit construire une phrase à travers différents choix d'expressions et de règles dans des menus.

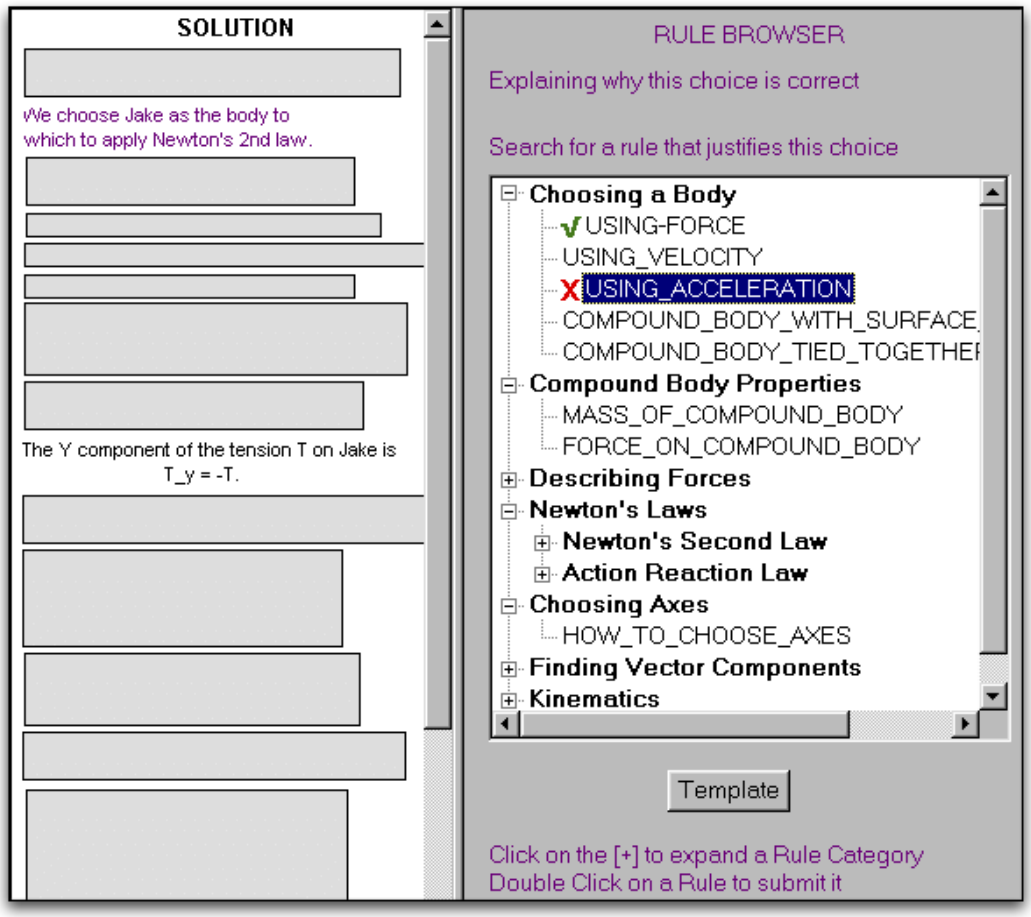


FIG. *3.1 – Le navigateur de règles dans SE-Coach (tiré de [Conati et VanLehn, 2000])*

Les résultats d'une évaluation empirique de SE-Coach montrent que l'échafaudage structuré de l'auto-explication peut être plus avantageux dans les premières étapes de l'apprentissage. D'après les auteurs, des formes plus simples d'incitation peuvent déclencher avec succès l'auto-explication lorsque les apprenants deviennent plus compétents dans le domaine.

KERMIT-SE est une version du système KERMIT (présenté dans le chapitre 2, paragraphe 2.4.1.4, page 35) qui intègre l'auto-explication [Weerasinghe et Mitrovic, 2002, Weerasinghe et Mitrovic, 2006]. Il ne s'agit pas pour l'apprenant de justifier chaque étape de modélisation, mais de donner une explication lorsqu'une contrainte est violée. L'explication prend la forme d'un dialogue tutoriel. Pour les erreurs simples, il se limite à un message de rétroaction détaillé. Pour les erreurs plus complexes, il consiste en une série de questions. Les questions peuvent être des questions fermées (avec réponse « oui / non » ou avec plusieurs propositions de réponses) ou des phrases comportant des blancs à compléter (cf. figure 3.2).

Etant donné que chaque dialogue tutoriel commence par expliquer l'erreur commise par l'apprenant, les auteurs expliquent que cela lui donne l'opportunité de réfléchir sur sa procédure de résolution du problème. Une évaluation de KERMIT-SE a montré que l'utilisation du système d'auto-explication améliore les capacités des apprenants à résoudre des problèmes et à répondre à des questions sur la connaissance du domaine.

Tutor-1: You've modelled *Grade* as an entity. What does that mean in terms of properties of an entity
Student-1: I don't know
Tutor-2: Let's try a simple thing. How do we model characteristics of an entity ?
Student-2 : Using Attributes
Tutor-3 : Great ! Nowtell me whymodelling *Grade* as an entity contradicts with the problem description ?
Student-3 : I'm not sure
Tutor-4: Don't worry. I'm here to help you. Check carefully whether you can find any attributes for *Grade*.
{After sometime} Pls. tell me whether you manage to find any attributes for *Grade*
Student-4: No
Tutor-5: Fine :It doesn't have any attributes. So now can you tell me why modelling *Grade* as an entity contradicts with the problem description ?
Student -5: Because it doesn't have any attributes to describe its characteristics
Tutor-6: Great! Now tell me the best way to model *Grade*?
Student-6 : I need more help
Tutor-7: Ok. If *Grade* doesn't have attributes, it cannot be an entity. So it should either be a relationship or an attribute. Now tell me the best way to model *Grade*.
Student-7 : Attribute
Tutor-8 : Great ! Now use the correct word to fill in the blanks so that you'll remember what you learnt.
…….. are used to describe characteristics of an …….. in a ER model
Student-8 : attributes, entity
Tutor-9: Excellent, Go ahead and do the changes. Submit the solution when you're done

FIG. *3.2 – Un dialogue tutoriel simple dans KERMIT-SE (tiré de [Weerasinghe et Mitrovic, 2002])*

Auto-évaluation

[Roll et al., 2006] ont développé un tuteur pour l'auto-évaluation : **Self-Assessment Tutor** (SAT). Intégré au tuteur *Geometry Cognitive Tutor* pour l'apprentissage de la géométrie, SAT a pour objectif d'encourager l'auto-évaluation spontanée. Pour cela, une activité préparatoire d'auto-évaluation est ajoutée avant chaque ensemble de problèmes traitant une compétence donnée (cf. figure 3.3). Un problème relevant de cette compétence est proposé à l'apprenant et celui-ci peut répondre à quatre types de questions :

- *Prédiction* : l'apprenant peut prédire s'il saura ou non résoudre le problème ;
- *Tentative* : l'apprenant propose une réponse au problème ;
- *Réflexion* : les questions sont par exemple : « Ai-je réussi ? » ou « Est-ce conforme à ma prévision ? » ;
- *Projection* : les questions concernent l'utilisation future de la compétence visée. Par exemple : « Qu'est ce que cela implique concernant ma capacité à résoudre dans le futur des problèmes portant sur la même compétence ? », « Aurai-je besoin d'aide la prochaine fois que j'aurai à résoudre un problème qui requiert cette compétence ? ».

L'auto-évaluation est également utilisée dans l'approche de [Amado Gama, 2004] L'auteur propose un modèle nommé « Reflection Assistant Model » pour favoriser de manière explicite les compétences métacognitives de l'apprenant dans le contexte de la résolution de problèmes en définissant les compétences métacognitives spécifiquement visées, les moments pour travailler ces compétences, les approches appropriées pour développer ces compétences et les mécanismes pour évaluer les changements métacognitifs des apprenants.

Dans cette approche, la réflexivité proposée par le système sur les processus d'apprentissage, les capacités de l'apprenant et sur l'activité réalisée est vue comme un moyen d'améliorer les compétences métacognitives de l'apprenant.

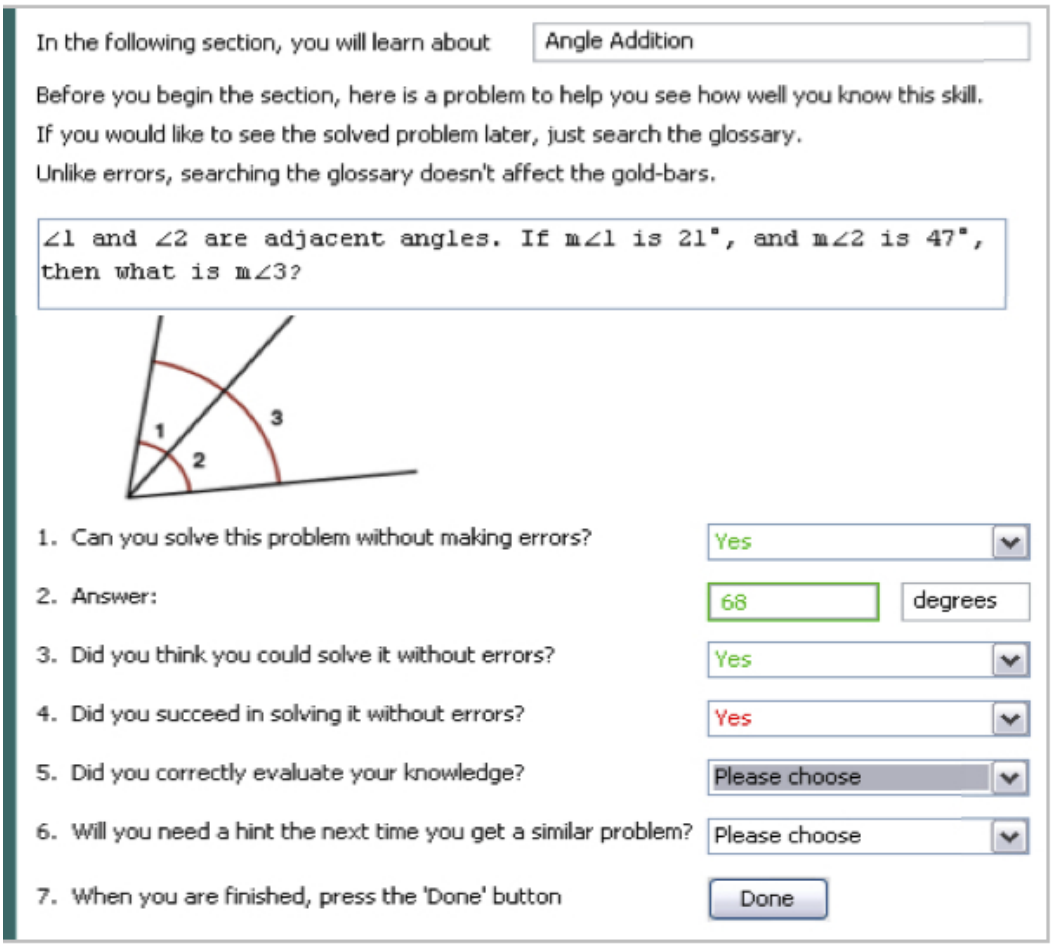


FIG. *3.3 – Un exercice d'auto-évaluation dans Self-Assessment Tutor (tiré de [Roll et al., 2006])*

Le modèle se focalise sur les compétences métacognitives suivantes :

- la surveillance de la connaissance (*knowledge monitoring*). L'échafaudage de la surveillance de la connaissance est l'objectif principal du modèle. Pour y parvenir l'assistant construit automatiquement un profil métacognitif de l'apprenant en se basant sur deux mesures :
  - *Knowledge Monitoring Accuracy* (KMA) : qui mesure la justesse de l'estimation de ses connaissances par l'apprenant.
  - *Knowledge Monitoring Bias* (KMB) : qui mesure la capacité de l'apprenant à contrôler et surveiller ses connaissances.
- l'évaluation de l'apprentissage : les activités proposées dans le modèle ont pour objectif de développer la prise de conscience par l'apprenant de son comportement lors de la résolution de problèmes, des ressources qu'il utilise, du temps passé sur chaque tâche, et des décisions prises au cours du processus de résolution de problème.
- la sélection de stratégies métacognitives. Le modèle se concentre sur le développement de la prise de conscience par l'apprenant des stratégies de contrôle de la compréhension, de contrôle du processus de résolution de problèmes et sur les stratégies de révision.

Le modèle a été implémenté et testé dans MIRA, un environnement pour l'apprentissage de problèmes algébriques. La figure 3.4 montre la réification graphique des indicateurs KMA et KMB.

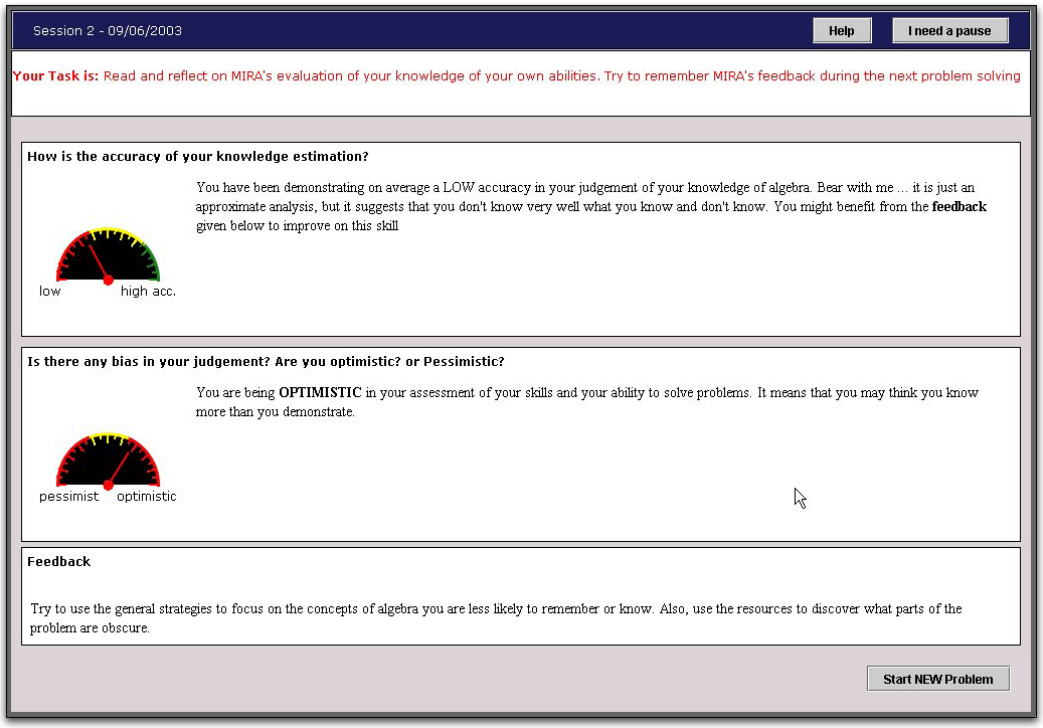


FIG. *3.4 – Analyse de la surveillance de la connaissance dans MIRA (visualisation des indicateurs KMA et KMB). L'apprenant observe l'interprétation de ses compétences métacognitives et lit le message de rétroaction (tiré de [Amado Gama, 2004]).*

Une expérimentation a montré que les apprenants utilisant la version de MIRA intégrant le modèle passaient plus de temps sur les problèmes et abandonnaient moins souvent que les apprenants du groupe de contrôle (ayant travaillé avec une version de MIRA sans le support à la métacognition). L'utilisation de l'assistant a un impact positif sur la métacognition des apprenants et sur les processus d'apprentissage.

Guidage métacognitif pour l'autorégulation

[Sakdavong et al., 2008a] s'intéressent au guidage métacognitif de l'apprenant dans les EIAH. Ces travaux se situent dans le cadre du projet ANR CEAMATIC dont l'objectif est de suporter l'autorégulation de l'apprenant en mettant en place un système d'aide dynamique basé sur l'analyse en temps réel de son activité. Dans ce contexte, les auteurs ont implémenté le dispositif ACMAMA (Architecture Cognitive Multi-Agents pour un Meilleur Apprentissage). Ce système permet de capturer et d'analyser l'activité de l'apprenant et de réagir en proposant ou imposant des aides cognitives ou métacognitives. L'une de ses particularités est de comparer plusieurs types d'aide afin de proposer ou d'imposer les aides les plus pertinentes en fonction des actions précédentes et du profil de l'apprenant. Ce dispositif a été expérimenté dans le cadre de l'apprentissage d'un traitement de texte [Sakdavong et al., 2008b] et les résultats sont en cours d'analyse.

Collaboration

Les environnements collaboratifs ont pour but de favoriser les interactions entre apprenants. Ils sont donc naturellement utilisés dans de nombreuses études pour développer les compétences métacognitives.

Dans certains systèmes, la collaboration se fait par le biais d'un agent pédagogique qui joue le rôle d'un compagnon d'apprentissage (*learning companion*) pour l'apprenant. Ils apportent un regard critique sur le travail de l'apprenant et favorisent l'auto-réflexion. D'après [Goodman et al., 1997] les agents pédagogiques ont l'avantage de pouvoir s'adapter aux connaissances et aux besoins de l'apprenant pour être plus efficace – contrairement aux collaborateurs humains. L'assistant dans [Tholander et al., 1999] se présente sous la forme d'un perroquet (cf. chapitre 2, figure 2.20, page 41). Les objectifs de cette approche visent à améliorer le *monitoring* et la régulation métacognitive au cours de la tâche de modélisation. Pour cela l'agent animé pose des questions qui ont pour but d'amener l'apprenant à réfléchir sur son diagramme et d'encourager ainsi l'auto-réflexion (cf. chapitre 2, paragraphe 2.4.2.3, page 40 pour plus de détails).

D'autres environnements utilisent la collaboration entre pairs pour favoriser la métacognition et l'apprentissage. Pour permettre une interaction efficace, l'environnement collabaratif doit mettre à disposition des apprenants des outils de communication, de coordination et de métacognition dans un espace d'activité commun [Romero, 2003]. Dans la version collaborative de Collect-UML [Baghaei et Mitrovic, 2006], les apprenants disposent d'un espace de travail privé, dans lequel ils construisent leur propre solution, et d'un espace de travail partagé pour la construction d'une solution collective. Seul un apprenant à la fois peut modifier la solution commune. Les apprenants disposent aussi d'un *chat* pour échanger leurs idées. Le système peut proposer des rétroactions concernant la tâche de modélisation ou fournir des conseils pour encourager la discussion entre apprenants.

La collaboration facilite également les compétences d'organisation et de planification du travail collaboratif des membres engagés dans l'activité. MIST (*Metacognition In Studying from Texts*) [Puntambekar et du Boulay, 1999] est un environnement pour aider les apprenants à développer une approche systématique de l'apprentissage à partir de textes en déployant tout un ensemble d'activités de planification et de suivi. L'accent est mis sur la prise de conscience du processus d'apprentissage plutôt que sur l'objet de l'apprentissage. Les apprenants travaillent avec MIST en binôme. Les questions du système et les activités proposées ont pour objectif de générer la collabaration pour une réflexion commune sur les compétences métacognitives mises en jeu lors de l'apprentissage à partir de textes. MIST donne des instructions explicites sur la façon dont chaque apprenant peut aider l'autre en se basant sur les activités et choix de chacun.

3.4 Notre approche : soutenir l'activité réflexive et métacognitive durant l'activité de modélisation

Nous retenons de ce chapitre les éléments suivants :

– Acquérir des capacités métacognitives, par exemple la capacité de se réguler, de s'auto-évaluer, ou de réfléchir sur ses actions, est favorable à l'apprentissage. Dans le cadre de l'apprentissage de la modélisation orientée objet, maîtriser des compétences métacognitives est d'autant plus important que la tâche de modélisation est une tâche ouverte : l'apprenant doit réguler son activité et son apprentissage au cours de la

tâche de modélisation. Ainsi, des compétences telles que se poser des questions, planifier ses activités, contrôler le résultat de ses activités, s'évaluer constamment avant, pendant et après une tâche et se réajuster si besoin sont essentielles pour la tâche de modélisation.

– Nous souhaitons proposer des moyens d'interaction pour permettre aux apprenants de mobiliser des processus métacognitifs. A travers l'interaction dans l'EIAH nous souhaitons donc :
  • favoriser l'activité réflexive et métacognitive de l'apprenant par rapport à son travail ;
  • amener l'apprenant à se questionner sur le modèle contruit, favoriser son autonomie. Pour cela, nous souhaitons aider l'apprenant à comprendre le modèle qu'il propose par l'explicitation de concepts implicites dans les notations graphiques utilisées et par la reformulation des éléments graphiques du modèle.
– Concernant l'intégration de la métacognition dans les EIAH, nous avons présenté la classification en trois dimensions (temps, niveau, approche) de [Amado Gama, 2004], les 10 principes de prise en compte du développement métacognitif dans les EIAH de [Romero, 2005] ainsi que quelques exemples d'EIAH intégrant une dimension métacognitive. De nombreux travaux concernent le domaine de la résolution de problèmes et explicitent les connaissances métacognitives visées comme la recherche d'une stratégie de résolution, la vérification de son efficacité, l'évaluation des capacités de l'apprenant à résoudre une tâche donnée, etc. L'explicitation se fait par des activités dédiées, en particulier avant et après la tâche de résolution. Cette approche est difficilement transposable à l'apprentissage de la modélisation orientée objet car on ne peut pas évaluer un pas de l'activité de modélisation, contrairement à un pas d'une résolution de problème ou de preuve. Notre choix porte sur l'intégration d'aides et de rétroactions permettant d'exercer les capacités de régulation métacognitive de l'apprenant. Concernant la dimension de temps, nous optons pour des aides au cours de l'activité de modélisation pour permettre un contrôle pendant la modélisation, ou après l'activité pour viser l'évaluation du résultat de l'activité en question. Dans notre contexte, le travail métacognitif est effectué en plus de l'activité principale, qui reste l'activité de modélisation. Nous ne travaillons pas sur des compétences métacognitives génériques mais nous souhaitons favoriser les compétences métacognitives liées à l'activité de modélisation telles que l'auto-contrôle et les fonctions de régulation. Enfin, nous ne nous limitons pas à une seule approche pédagogique en particulier : nos propositions pourront utiliser différents mécanismes, approches ou outils afin d'encourager l'activité métacognitive.

Le chapitre suivant décrit notre proposition d'un modèle d'interaction pour un EIAH d'apprentissage de la modélisation. Un des objectifs visés par ce modèle est de favoriser l'activité métacognitive de l'apprenant.

CHAPITRE 4

Un modèle d'interaction pour un EIAH d'apprentissage de la modélisation orientée objet

Sommaire

Résumé

Ce chapitre est consacré au modèle d'interaction pour l'apprentissage de la modélisation que nous proposons. Ce modèle est générique et peut être instancié à différents domaines de la modélisation informatique comme la modélisation orientée objet ou les bases de données. Il est formé de quatre composants : les tâches que nous proposons aux apprenants, les modes d'interaction dans l'environnement d'apprentissage, les aides métacognitives et les rétroactions à apporter à l'apprenant. Dans un premier temps, nous présentons l'aspect générique de ce modèle, et dans un second temps nous détaillons son application à la modélisation par diagramme de classes qui constitue un aspect essentiel de l'apprentissage de la modélisation orientée objet.

4.1 Proposition d'un modèle générique d'interaction pour l'apprentissage de la modélisation informatique

Ce paragraphe s'organise en quatre parties selon les quatre dimensions de notre modèle : les **tâches et activités d'apprentissage** à mettre en œuvre, les **modes d'interaction** entre l'environnement et l'apprenant, les **aides métacognitives** au cours de l'activité et les **rétroactions** du système.

4.1.1 Tâches et activités d'apprentissage de la modélisation informatique

Nous décrivons ici un ensemble de **tâches** génériques pour l'apprentissage de la modélisation informatique. Ces tâches sont décrites à un niveau général, sans méthode pour les aborder. Nous proposons ensuite d'utiliser une méthodologie en trois étapes, que nous appliquons à chaque tâche : nous obtenons alors ce que nous nommerons des **activités**.

4.1.1.1 Tâches d'apprentissage de la modélisation

Dans un premier temps, nous avons étudié différents ouvrages dédiés à l'initiation à la modélisation en informatique ainsi que plusieurs corpus d'exercices de modélisation informatique à destination d'apprenants novices afin de déterminer quels types de tâches nous pouvions proposer. Nous avons identifié trois types de tâches applicables à différents domaines de la modélisation informatique. Chacune des tâches que nous proposons se base sur une description textuelle du problème comme de nombreux ouvrages le préconisent (cf. chapitre 2, paragraphe 2.3.1.1, page 25). La tâche de « **modèle à créer** » est la tâche que nous retrouvons le plus souvent dans les exercices de modélisation. Elle consiste à élaborer entièrement le modèle à partir de l'énoncé du problème. Certains exercices se basent sur une ébauche du modèle que l'apprenant doit compléter : nous proposons une tâche « **modèle à compléter** » correspondant à ce type d'exercice. Enfin, certains exercices proposent un modèle dont l'apprenant doit discuter les choix de modélisation. Nous proposons donc une tâche de « **modèle à corriger** » dans laquelle l'apprenant doit retrouver des erreurs de modélisation dans un modèle donné.

Dans un second temps, nous avons déterminé, pour chaque type de tâche, l'intérêt et les difficultés propres à chacunes d'entre elles ainsi que les compétences nécessaires et visées par la tâche.

La tâche « modèle à créer » est la tâche usuelle en modélisation, celle que les apprenants rencontrent dans la majorité des situations de modélisation, que ce soit dans le cadre

scolaire ou professionnel. Cette tâche exerce la créativité de l'apprenant car il ne dispose que de l'énoncé pour contruire son modèle.

La tâche « modèle à compléter » est un autre type de tâche, que l'on rencontre moins fréquemment dans les enseignements en modélisation. Dans ce type d'exercice, l'apprenant doit principalement repérer les éléments qui permettent de compléter le modèle. Si ce type de tâche semble moins complet que l'activité de création, il requiert d'autres compétences comme l'aptitude d'analyse de l'ébauche de solution donnée ou le repérage d'éléments manquants lorsqu'ils ne sont pas indiqués dans la consigne. Ce type d'exercice peut permettre de centrer le travail de l'apprenant sur une difficulté ou un concept particulier du domaine étudié en éliminant les autres difficultés de l'exercice. Pour construire l'ébauche de modèle, il peut être intéressant de se baser sur un modèle de référence dont on supprime certains éléments.

La dernière tâche que nous avons identifiée est la tâche « modèle à corriger ». Pour corriger un modèle donné, l'apprenant doit comprendre tout l'énoncé ainsi que le modèle proposé. Les exercices correspondant à ce type de tâche ne sont pas des exercices de création mais sont davantage centrés sur des choix de modélisation à étudier ou des notions de modélisation elles-mêmes afin, par exemple, de vérifier leur acquisition. Ce type de tâche permet d'exercer le sens critique nécessaire à l'auto-correction et détient donc une fonction d'évaluation au sens de la régulation métacognitive. Un exemple d'exercice de ce type peut être de proposer à un étudiant de critiquer et corriger le modèle réalisé par un autre étudiant, ou de demander à un étudiant de corriger un modèle qui comporte des erreurs « classiques » du domaine étudié. Dans ce cas, pour construire le modèle erroné, on peut se baser sur un diagramme de référence dont on modifie certains éléments. Un cas particulier de cette tâche serait de donner un modèle à la fois à corriger et à compléter.

4.1.1.2 Utilisation d'une méthodologie en trois phases

Bien qu'il n'existe pas de méthode de modélisation unique pour la construction d'un modèle informatique, il nous paraît important de fournir aux apprenants une méthode générale pour aborder un problème de modélisation. Nous préconisons donc une approche en trois étapes, mise au point par Thierry Lemeunier, sur la base de son expérience d'enseignant :

- **Etape de lecture** : la première étape consiste à lire entièrement l'énoncé pour se l'approprier. Lors de cette étape, l'apprenant découvre le sujet général de l'énoncé et repère les concepts importants.
- **Etape de modélisation** : c'est la phase d'élaboration du modèle correspondant à l'énoncé.
- **Etape de relecture** : la troisième étape consiste à relire l'énoncé de l'exercice et à le comparer au modèle construit afin de vérifier que le modèle est correct et complet par rapport à l'énoncé.

L'organisation générale en trois étapes permet à l'apprenant de planifier ses activités et la dernière étape est un moment privilégié pour favoriser la réflexion de l'apprenant sur son travail et favoriser l'auto-correction. C'est pourquoi nous avons choisi d'utiliser cette méthodologie comme base à l'ensemble des activités pour l'apprentissage de la modélisation informatique que nous proposons.

4.1.1.3 Proposition d'activités

Pour chaque type de tâche, nous proposons une activité structurée en trois phases suivant la méthode décrite dans le paragraphe précédent.

Dans l'activité « modèle à créer », l'apprenant doit construire entièrement le modèle à partir d'un énoncé textuel. Il est assez naturel d'organiser cette activité autour des trois phases de lecture, modélisation et relecture car la méthode de modélisation en trois phases a principalement été réalisée pour ce type d'activités.

Dans l'activité « modèle à compléter », l'apprenant doit compléter une ébauche de solution donnée avec l'énoncé. La première étape de cette activité diffère donc de celle de l'activité « modèle à créer » car il s'agit non seulement de lire l'énoncé mais également d'analyser l'ébauche de solution proposée et de repérer les éléments manquants. La deuxième étape consiste ensuite à compléter le modèle donné avec les éléments repérés dans la première étape. La dernière étape consistera, comme pour l'activité de « modèle à créer », à vérifier que le modèle obtenu est complet et correct par rapport à l'énoncé.

Dans l'activité « modèle à corriger » l'apprenant doit corriger un modèle dans lequel des erreurs ont été introduites. La première étape de cette activité diffère de celle des deux activités précédentes : il s'agit dans ce cas de lire l'énoncé et d'analyser l'ébauche de solution proposée afin de repérer les erreurs de modélisation. Dans la deuxième étape l'apprenant doit alors corriger les erreurs repérées dans la première phase. La dernière étape consiste, comme dans les autres types d'activités, à vérifier la correction et la complétude du modèle obtenu par rapport à l'énoncé.

Ces activités peuvent être réalisées dans un environnement papier/crayon ou dans un environnement informatique, comme c'est le cas dans notre étude. Nous proposons pour cela d'intégrer dans l'EIAH des modes d'interaction particuliers.

4.1.2 Modes d'interaction

L'intégration dans un EIAH des activités présentées dans le paragraphe précédent nécessite de mettre en œuvre différents modes d'interaction entre les apprenants et l'environnement informatique. Pour y parvenir nous proposons trois éléments : en premier lieu, nous intégrons la description textuelle de l'exercice à l'interface de l'EIAH. Nous proposons ensuite une réification des activités au sein de l'environnement. Enfin, nous ajoutons dans l'environnement un ensemble d'outils graphiques pour la modélisation, exploitant la présence de l'énoncé à l'interface.

4.1.2.1 Intégration de l'énoncé à l'interface

Les activités de modélisation informatique que nous proposons se basent sur un énoncé textuel. Lors d'un travail en environnement papier/crayon, l'énoncé est fourni sous forme papier à l'apprenant. Pour la réalisation d'activités dans un environnement informatique d'apprentissage, nous proposons d'intégrer l'énoncé à l'interface, comme le font les environnements d'apprentissage de la modélisation KERMIT et Collect-UML (présentés dans le chapitre 2, paragraphes 2.4.1.4 et 2.4.2.2, pages 35 et 38). Avec un support papier de l'énoncé, effectuer des allers et retours entre l'énoncé et le modèle construit dans l'environnement n'est pas une tâche aisée. L'affichage à l'interface à la fois du texte de l'énoncé et du modèle facilite ce contrôle visuel car l'apprenant ne dispose plus que d'un unique support.

L'intégration de l'énoncé à l'interface permet également d'ajouter des outils graphiques de modélisation et augmente ainsi les possibilités d'interaction dans l'environnement par rapport à un éditeur UML classique. Dans KERMIT et Collect-UML par exemple, les éléments du modèle sont créés à partir d'éléments de l'énoncé et un code de couleur est utilisé pour chaque type d'élément et pour le mot de l'énoncé correspondant. De plus, les apprenants utilisent des outils graphiques en mode papier/crayon : ils peuvent souligner, barrer, surligner des expressions de l'énoncé. Ajouter le texte à l'interface permet d'ajouter des outils graphiques de modélisation dans l'environnement et ainsi de recréer les outils dont les apprenants disposent en mode papier/crayon.

4.1.2.2 Scénarisation de l'interaction

La scénarisation globale des activités est basée sur les trois étapes (lecture/modélisation/relecture) de la méthodologie présentée dans le paragraphe 4.1.1. La figure 4.1 montre l'enchaînement de ces étapes.

FIG. *4.1 – Enchaînement des étapes pour chaque activité*

Le passage de la première étape à la deuxième est irréversible : lorsque l'apprenant estime avoir atteint les objectifs de cette étape, il peut passer à la deuxième étape. La modélisation est une activité itérative : des allers et retours entre les étapes de modélisation et de vérification sont autorisées, jusqu'à ce que l'apprenant soit satisfait du modèle final.

Les objectifs de la première étape diffèrent selon l'activité proposée. Pour l'activité de « modèle à créer », l'énoncé est le seul élément dont dispose l'apprenant au commencement de l'activité : l'interface ne comprend donc que l'énoncé au cours de cette phase. Pour les activités de « modèle à compléter » ou « à corriger » en revanche, un espace de modélisation graphique est ajouté à l'interface de la première étape, car l'apprenant doit analyser le modèle proposé. Cet espace permet de visualiser – mais pas de modifier – l'ébauche de diagramme ou le diagramme à corriger.

L'étape de modélisation s'organise de la même façon pour toutes les activités : l'apprenant dispose de l'énoncé et d'un espace de modélisation graphique dans lequel le modèle est modifiable.

Durant la dernière étape, l'apprenant doit relire l'énoncé de l'exercice et le comparer à son modèle afin de vérifier qu'il est correct et complet par rapport à l'énoncé. L'interface comprend donc à la fois l'énoncé et l'espace de modélisation graphique dans lequel le modèle n'est pas modifiable.

4.1.2.3 Outils graphiques de modélisation

Nous proposons un ensemble d'outils graphiques exploitant l'affichage de l'énoncé à l'interface de l'EIAH en fonction des différentes étapes de la scénarisation.

Quelle que soit l'activité, l'apprenant doit, dans la première phase, lire l'énoncé et repérer les éléments importants. Lors d'un travail avec un support papier, les apprenants peuvent utiliser un crayon ou un surligneur pour marquer les mots qui leur semblent

pertinents pour la modélisation. Nous proposons un **outil de soulignage** de mots ou d'expressions de l'énoncé tel qu'il en existe dans les éditeurs de textes classiques. Cela permet à l'apprenant de mettre en évidence les expressions importantes qu'il a repérées dans l'énoncé et d'accèder alors à un outil similaire à celui dont il dispose en mode papier/crayon.

Pour la création d'éléments graphiques, nous proposons un **outil de coloriage** qui permet de créer directement un élément du modèle à partir d'une expression de l'énoncé. Cet outil est comparable à celui existant dans KERMIT et Collect-UML mais diffère en deux points : dans KERMIT et Collect-UML, tous les éléments d'un type donné dans le modèle sont affichés de la même couleur alors que nous proposons d'utiliser une couleur par expression. L'expression sélectionnée et l'élément graphique sont affichés dans une couleur identique, afin de faciliter le contrôle visuel. De plus, dans KERMIT et Collect-UML la création d'éléments graphiques est possible uniquement avec l'utilisation d'expressions de l'énoncé. Les énoncés doivent donc décrire de façon exhaustive tous les éléments du modèle ce qui limite les possibilités de varier le niveau de détail de l'énoncé. En effet, certaines descriptions peuvent être incomplètes ou bien elles n'explicitent pas la totalité des éléments du modèle. Dans ce cas, il est difficile d'utiliser l'énoncé pour créer des éléments. C'est pour cela que nous proposons d'utiliser deux modes d'interaction dans l'EIAH : un **mode libre** et un **mode assisté**. Le mode assisté exige que l'apprenant utilise l'outil de coloriage pour créer les éléments du modèle alors que le mode libre permet la création d'éléments sans lien avec l'énoncé. Tout élément créé en mode libre peut être relié ultérieurement à l'énoncé grâce à l'**outil de rattachement**. Cet outil peut également être utilisé pour relier une expression de l'énoncé à une autre, par exemple les différentes occurences d'une expression ou d'un mot dans l'énoncé ou pour lier un mot à son synonyme. Les deux expressions prennent alors la même couleur, matérialisant ainsi le lien logique entre elles.

Durant la dernière étape, l'apprenant doit relire l'énoncé de l'exercice et le comparer à son modèle afin de vérifier qu'il est correct et complet par rapport à l'énoncé. Nous proposons pour cette phase un outil particulier : **l'outil de passage de la souris** qui facilite cette vérification. Au début de cette phase, l'interface ne présente plus que l'énoncé, en noir et blanc. L'apprenant doit alors parcourir le texte avec sa souris en suivant l'ordre des mots. Lorsque la souris survole une expression modélisée, celle-ci reprend sa couleur et les éléments correspondants dans le modèle réapparaissent simultanément à l'interface.

Pour les activité de correction et complétion de modèle, nous proposons d'utiliser également l'outil de passage de la souris dans la première phase pour faciliter l'analyse du modèle existant. Pour cela, nous supposons que les éléments du modèle initial sont liés à l'énoncé lorsque l'apprenant commence son activité.

4.1.3 Aides métacognitives au cours de la modélisation

Notre modèle d'interaction est également basé sur la proposition d'aides à l'apprenant au cours de la modélisation. Nous avons montré dans le chapitre 3 qu'il est nécessaire de favoriser l'utilisation des fonctions de régulation métacognitive. Nous proposons donc deux aides métacognitives qui, mises en œuvre au cours de l'activité de modélisation, ont pour objectif d'encourager l'apprenant à contrôler et à évaluer son activité dans l'EIAH. Ces aides nous ont été inspirées par l'observation de séances de travaux pratiques de modélisation orientée objet mais ont un caractère générique. Il s'agit de l'aide à la création d'éléments graphiques et la reformulation d'éléments graphiques.

4.1.3.1 Aide à la création d'éléments graphiques

Au cours d'observations de l'utilisation par les apprenants de différents environnements de modélisation, nous avons constaté que les étudiants avaient parfois des difficultés à utiliser les fonctionnalités du logiciel et n'associaient pas de sémantique à leurs actions. Par exemple, dans certains domaines il existe des relations orientées. Lors de la création de telles relations dans un environnement informatique, l'ordre dans lequel deux éléments à relier sont sélectionnés détermine l'orientation de la relation. Les apprenants ne sont pas toujours conscients de cette contrainte, absente sur le papier, et ceci est la source de nombreuses erreurs. Un autre cas est l'utilisation d'un mauvais type de relation (par exemple deux relations dont la représentation graphique est proche mais dont la signification est très différente).

Pour les aider à surmonter ces difficultés nous proposons dans notre modèle d'interaction un guidage contextuel lors de la création d'éléments. Ce dispositif s'apparente aux rétroactions produites dans le logiciel Cabri-Géomètre lors d'opérations élémentaires [Laborde et Laborde, 2006]. D'après ces auteurs, « *ces messages soutiennent l'utilisateur dans l'usage du logiciel* » et « *restreignent la difficulté de la tâche* ». Le guidage que nous proposons se présente sous la forme d'un message indiquant quel élément est en cours de création. Le message est affiché durant l'activité de modélisation (c'est-à-dire durant la deuxième phase) et au cours de l'opération de création d'un élément. De plus, il est sensible au contexte : il s'adapte aux actions de l'apprenant tel que le changement de sélection d'élément, de type de relation, etc. Cela a pour objectif de faciliter le contrôle de l'activité par l'apprenant lui-même, et ainsi d'éviter les erreurs de construction en cours d'action.

4.1.3.2 Reformulation des éléments graphiques

Chaque domaine de modélisation en informatique se base sur des éléments de modélisation et leur sémantique. Au cours d'observations de l'utilisation par les apprenants de différents environnements de modélisation, nous avons constaté que la sémantique véhiculée par ces éléments n'est pas toujours perçue par les apprenants qui débutent. Par exemple, nous avons constaté qu'ils n'utilisent pas les mots du vocabulaire précis du domaine mais les remplacent par des mots comme « trait » au lieu de relation.

Nous faisons l'hypothèse que confronter l'apprenant à une reformulation textuelle de son modèle peut l'aider à comprendre sa signification et donc lui permettre de valider ou d'invalider ses constructions. Pour ce faire, nous proposons d'afficher une reformulation locale des principaux éléments du modèle de l'apprenant. Le message de reformulation s'affiche lorsque la souris passe sur un objet ou un groupe d'objets du modèle. Il précise le type des éléments et la reformulation usuelle associée à cet élément. Cela peut être particulièrement intéressant dans le cas de relations orientées. La reformulation de la relation met en avant le sens dans lequel cette relation se lit (en fonction de la sémantique des éléments du domaine). La reformulation sollicite donc la fonction de contrôle de l'activité.

4.1.4 Rétroactions

4.1.4.1 Rétroactions dans les EIAH

La conception des rétroactions est une tâche complexe qui dépend de façon intrinsèque de l'apprentissage visé par l'environnement informatique. Ainsi, de multiples facteurs tels que la nature de l'EIAH, les stratégies et le style pédagogique mis en œuvre dans l'EIAH

influencent les choix de conception et d'organisation des rétroactions. Nous retenons pour notre étude quatre axes généraux permettant de classifier les rétroactions d'un EIAH : les **objectifs** des rétroactions, le **moment d'intervention** du système, le **contenu** et la **forme** des rétroactions.

Les objectifs des rétroactions sont fortement liés aux objectifs pédagogiques de l'EIAH, eux-mêmes basés sur la nature de l'EIAH, le style pédagogique ainsi que les stratégies pédagogiques mises en œuvre. Ainsi, l'objectif des rétroactions dans un tuteur au style directif sera la correction d'une erreur identifiée à un pas précis de la résolution du problème. Les rétroactions peuvent aussi être de type méthodologique, en proposant à l'apprenant la meilleure marche à suivre pour résoudre un problème.

Le moment d'intervention du système correspond au moment où les rétroactions vont être données à l'apprenant et dépend du moment de validation permettant l'évaluation de la solution de l'apprenant et la construction des rétroactions. Dans un environnement laissant peu d'initiative à l'apprenant, les rétroactions peuvent être données dès qu'une erreur par rapport au plan prévu a été repérée. Dans les environnements de découverte guidée, les rétroactions ne sont pas déclenchées de façon systématique ou immédiate et ce choix donne une impression de liberté à l'apprenant.

Le contenu des rétroactions est lié aux objectifs et aux stratégies mises en œuvre dans l'environnement. Si l'objectif est de provoquer un conflit cognitif, le contenu de la rétroaction peut être un contre-exemple visant à déstabiliser l'apprenant afin qu'il remette en cause ses connaissances.

La forme des rétroactions fait référence à la présentation de celles-ci et varie selon les modalités d'interaction proposées dans l'environnement, comme le choix entre des modalités orales ou écrites par exemple. En outre, le support de la rétroaction peut être textuel ou graphique. Par ailleurs, de nombreux environnements informatiques s'attachent à utiliser le dialogue comme mode d'interaction comme par exemple [Michel, 2006] qui utilise des travaux en dialogue homme-machine pour la conception de l'interaction dans un EIAH pour l'apprentissage des langues.

4.1.4.2 Proposition de rétroactions

L'objectifs des rétroactions que nous proposons est d'amener l'appprenant à se poser des questions, à réfléchir à la pertinence de sa solution. Pour cela, nous faisons appel aux fonctions de contrôle et d'évaluation de la régulation métacognitive. En effet, l'absence de méthodes formelles de construction et de validation d'un modèle informatique nous amène à mettre l'accent sur l'acquisition par l'apprenant de procédures de contrôle, de correction et de validation de ses productions.

Le moment d'intervention dépend du moment de validation, c'est-à-dire du moment d'analyse du modèle réalisé par l'apprenant. Le moment d'analyse dépend de la méthode utilisée. Dans le cas d'un système à base de contraintes, la validation se fait à tout moment : dès qu'une contrainte est violée, le système réagit et fournit une rétroaction, comme dans des systèmes tels que KERMIT ou Collect-UML. Avec un système basé sur un outil de diagnostic, qui évalue le travail de l'apprenant, il faut que le modèle soit presque complet pour être analysé. Dans le cas de la modélisation, nous pensons qu'il est intéressant d'évaluer le travail de l'apprenant et de proposer des rétroactions une fois que l'apprenant juge son travail accompli. En effet, le processus de modélisation n'est pas linéaire et donner des rétroactions au cours de la modélisation limite l'apprenant car celui-ci peut souhaiter encore améliorer et affiner son modèle avant de demander une évaluation de son travail.

Dans la classification de [Amado Gama, 2004] présentée dans le chapitre 3, paragraphe 3.3.1.1, page 50, nous nous situons en ce qui concerne les rétroactions, dans une intégration de la métacognition après la tâche.

Le contenu des rétroactions doit être adapté au domaine de modélisation étudié afin que les rétroactions soient pertinentes pour l'apprenant. De plus, pour que celui-ci puisse évaluer sa solution, il faut que les rétroactions fassent référence à son propre travail.

Nous basons nos rétroactions sur une présentation écrite. Il s'agit d'intégrer des éléments métacognitifs tels que nous les avons décrits dans le chapitre 3, paragraphe 3.2.2, page 48, pour inciter l'apprenant à avoir un regard critique sur son diagramme et favoriser la régulation de son activité. Comme [Tholander et al., 1999] nous proposons des questions ou des commentaires relatifs au domaine de modélisation et en lien avec le modèle construit par l'apprenant.

En nous appuyant sur la classification proposée par [Lemeunier, 2000], nous avons retenu trois formes d'intervention, structurées de la plus générale à la plus directe : *indiquer*, *questionner*, *proposer*. « Indiquer » consiste à attirer l'attention de l'apprenant sur une partie de son modèle, ou sur la manière dont il a représenté une notion de l'énoncé. Cela peut consister en une reformulation textuelle d'un élément du modèle, 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 modèle. Les questions sont principalement de type binaire (réponse oui/non) et incitent l'apprenant à vérifier mentalement que le modèle satisfait une propriété donnée. Il s'agit dans ce cas d'encourager le processus métacognitif de vérification de son activité. L'apprenant est engagé dans un processus d'auto-questionnement (aucune réponse n'est exigée de la part de l'apprenant) et il apporte une réponse en vérifiant sa construction. La modalité « proposer », la plus directe, consiste à suggérer des modifications à apporter au modèle.

4.1.5 Rôle de l'enseignant dans la manipulation des concepts du modèle

Le modèle que nous proposons vise à être implémenté dans un environnement informatique pour l'apprentissage de la modélisation d'un domaine informatique. Ce logiciel pourra être utilisé lors de travaux pratiques d'initiation, en présence de l'enseignant et en complément des cours et travaux dirigés traditionnellement dispensés. L'enseignant joue donc un rôle important dans notre modèle puisqu'il intervient dans le choix des types d'activités et la rédaction de l'énoncé textuel. De plus, il intervient dans la conception du modèle à compléter et du modèle à corriger, en lien avec l'énoncé, pour les tâches de modèle à compléter et à corriger. Pour cela l'enseignant peut se baser sur un modèle de référence, qu'il doit également construire.

4.2 Application du modèle d'interaction à la modélisation orientée objet

Nous présentons dans ce paragraphe l'application du modèle générique, présenté dans le paragraphe précédent, au domaine de la modélisation orientée objet et à la modélisation par diagramme de classes en particulier. Nous avons retenu pour cela un sous-ensemble des éléments du langage UML utilisés pour la construction de diagrammes de classes que nous avons présentés dans le chapitre 2, paragraphe 2.2.3, page 19. Il s'agit des classes (classes, classes abstraites ou interfaces), des caractéristiques des classes (attributs ou opérations),

des relations (association, agrégation, composition, héritage, réalisation, dépendance) et des multiplicités associées aux associations, agrégations et compositions.

4.2.1 Tâches et activités d'apprentissage de la modélisation orientée objet

4.2.1.1 Tâches d'apprentissage de la modélisation orientée objet

Les trois tâches génériques que nous avons définies précédemment se déclinent pour la construction de diagrammes de classes en : « diagramme à créer », « diagramme à compléter » et « diagramme à corriger ». Nous avons étudié plusieurs manières de constituer un diagramme à compléter ou à corriger à partir d'un diagramme de référence, ainsi que l'intérêt pédagogique de chaque choix. Nous présentons ici notre analyse et des exemples de ces tâches, inspirés par des exercices de modélisation donnés par Thierry Lemeunier à ses étudiants.

La figure 4.2 présente un exemple d'énoncé appelé « Stylos et Feutres ». Cet énoncé seul peut servir de support à une tâche de diagramme à créer. Accompagné d'une ébauche de diagramme tel que le diagramme de gauche de la figure 4.3, il peut être utilisé pour une tâche de diagramme à compléter.

```
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 et un feutre sont
utilisés par une personne et appartiennent à une personne. Il
existe un feutre particulier qui est un feutre effaceur.
```

FIG. *4.2 – Enoncé « Stylos et Feutres »*

Pour élaborer un diagramme à compléter, nous pouvons par exemple supprimer toutes les relations du diagramme de référence, comme le montre la figure 4.3. Afin de résoudre cet exercice, l'apprenant doit repérer les concepts importants de l'énoncé. Ensuite, il doit repérer dans le squelette de solution quels concepts sont modélisés avec des classes puis déterminer les relations qui existent entre ces classes. Donner toutes les classes du diagramme réduit la difficulté de la tâche en particulier dans le cas de classes qui ne sont pas explicitement décrites dans l'énoncé. Cela évite les oublis de classes et les mauvaises représentations (par exemple un attribut au lieu d'une classe) et cela permet aussi de concentrer l'attention de l'apprenant sur le type des relations entre les classes, les multiplicités, etc. Une autre méthode pour créer un diagramme à compléter peut être de supprimer des éléments de chaque type. Dans ce cas, la difficulté peut être de s'adapter à la modélisation proposée, qui n'est pas nécessairement celle que l'apprenant aurait choisie. La difficulté de la création de l'ébauche de diagramme réside dans le choix du nombre et de la nature des éléments manquants.

L'objectif de la tâche de diagramme à corriger est d'inciter l'apprenant à contrôler les éléments du diagramme pour repérer d'éventuels problèmes de modélisation. Pour élaborer un diagramme à corriger, nous pouvons concentrer les erreurs sur des éléments précis comme le choix des classes, les multiplicités des relations ou le type de relation. Pour résoudre l'exercice, l'apprenant doit utiliser des capacités d'analyse de la solution présentée : il doit analyser chaque relation pour vérifier sa signification et sa pertinence dans le diagramme de classes.

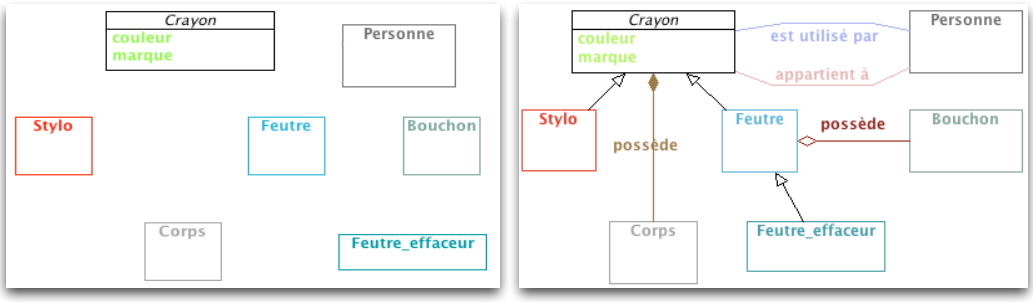


FIG. *4.3 – Ebauche de diagramme (à gauche) et diagramme de référence (à droite)*

Les erreurs peuvent porter sur les relations : par exemple le remplacement d'une relation par une autre, l'inversion du sens d'une relation comme l'illustre la figure 4.4.

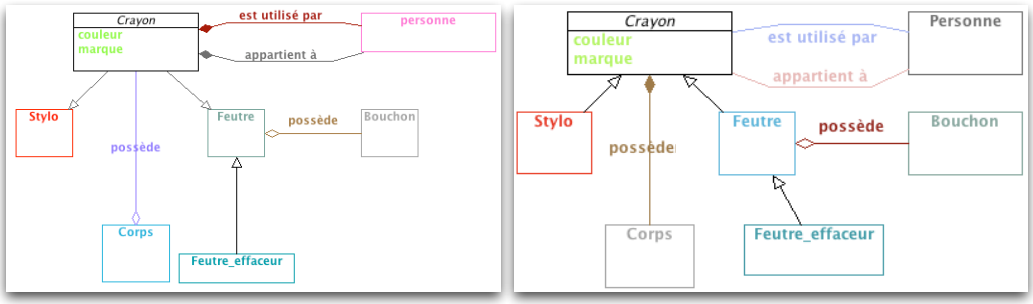


FIG. *4.4 – Diagramme à corriger (à gauche) et diagramme de référence (à droite)*

Cet exemple s'appuie sur l'énoncé « Stylos et Feutres » présenté précédemment. Les erreurs introduites sont les suivantes :

- le sens de l'héritage entre 'Stylo' et 'Feutre' est inversé ;
- le sens de l'héritage entre 'Feutre' et 'Feutre' est inversé ;
- la composition entre 'Crayon' et 'Corps' est remplacée par une agrégation dans le sens inverse ;
- l'association 'est utilisé par' entre 'Crayon' et 'Personne' est remplacée par une composition ;
- l'association 'appartient à' entre 'Crayon' et 'Personne' est remplacée par une composition ;

La difficulté de modélisation de ce type d'exercice est donc réduite car les classes sont déjà représentées. Cela permet à l'apprenant de se concentrer sur les relations, qui elles peuvent comporter des erreurs.

Une autre manière de construire le diagramme à corriger est d'introduire des erreurs sur tous les types d'éléments : représentation erronée d'une classe (remplacement par un attribut), transfert d'un attribut vers une autre classe, type erroné de relation, ou inversion du sens d'une relation. La résolution de cette tâche est complexe car il faut considérer l'ensemble du diagramme proposé et plus seulement un type d'élément.

Nous avons testé nos propositions en réalisant des exercices de ce type en collaboration avec Thierry Lemeunier. Ce travail a fait l'objet de mises à l'essai exploratoires en environnement informatique (cf. chapitre 6, paragraphe 6.1.2, page 6.1.2).

4.2.1.2 Application de la méthodologie en trois étapes

La méthodologie en trois étapes s'adapte sans difficulté à la modélisation orientée objet. Nous avons présenté dans le chapitre 2 paragraphe 2.3.1.1, page 25, les préconisations de différents ouvrages concernant l'apprentissage de la modélisation orientée objet comme par exemple l'identification des classes, attributs et relations à partir de la description textuelle du problème. La première étape de la méthodologie que nous proposons correspond à cette étape de repérage des éléments importants pour la modélisation du diagramme de classes. La deuxième phase est l'étape d'élaboration du diagramme, et la troisième étape consiste à vérifier la correction et la complétude du diagramme de classes par rapport à l'énoncé.

4.2.1.3 Activités d'apprentissage de la modélisation orientée objet

Les trois activités d'apprentissage de la modélisation par diagrammes de classes UML que nous proposons sont les trois tâches d'apprentissages de création, correction et complétion du diagramme de classes dont le déroulement suit les trois étapes de la méthodologie, comme nous l'avons défini dans le paragraphe 4.1.1, page 62. Ces activités sont les suivantes :

- « diagramme à créer » : la première phase consiste à lire l'énoncé et à repérer les éléments importants pour le diagramme de classes modélisé dans la deuxième phase. La troisième phase consiste à relire l'énoncé et à vérifier que le diagramme est correct et complet par rapport à l'énoncé.
- « diagramme à compléter » : la première phase consiste à lire l'énoncé et à analyser l'ébauche de diagramme de classes afin de repérer les éléments manquants. La deuxième étape consiste à compléter le diagramme de classes. La troisième étape consiste à vérifier que le diagramme est correct et complet par rapport à l'énoncé.
- « diagramme à corriger » : la première phase consiste à lire l'énoncé et à analyser le diagramme fourni afin de repérer les erreurs de modélisation. La deuxième étape consiste à corriger le diagramme de classes. La troisième étape consiste à vérifier que le diagramme est correct et complet par rapport à l'énoncé.

4.2.2 Modes d'interaction

Les modes d'interaction définis dans le paragraphe 4.1.2, page 64 s'appliquent sans difficulté à l'apprentissage de la modélisation orientée objet, et en particulier à la modélisation par diagramme de classes. L'énoncé décrivant le diagramme de classes est affiché à l'interface et l'interaction est scénarisée selon l'activité que l'apprenant doit réaliser. Celui-ci dispose des outils graphiques de modélisation. L'outil de soulignage permet par exemple à l'apprenant de repérer les noms qui correspondent aux classes ou aux attributs, et les verbes qui correspondent aux relations.

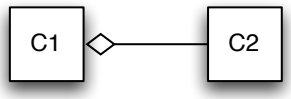
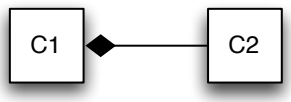
4.2.3 Aides métacognitives au cours de la modélisation

4.2.3.1 Aide à la création d'éléments graphiques

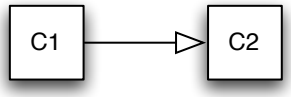
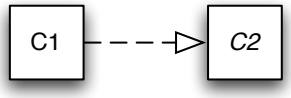
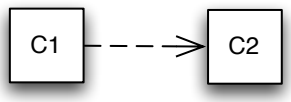
Nous proposons d'appliquer l'aide à la création à tous les types de classes, aux attributs et opérations et à toutes les relations du diagramme. Nous avons défini un ensemble de messages, s'affichant à différents moments au cours de la création d'un élément que nous présentons dans le tableau 4.1.

Concernant les classes, classes abstraites et interfaces, le message a pour objectif d'indiquer l'action en cours avant que l'action de création ne soit finalisée afin de faciliter le contrôle par l'apprenant. Le message indique le type de la classe en cours de création. Pour les attributs et les opérations, il s'agit de reformuler la signification du résultat de l'activité. Si un attribut A est ajouté dans une classe C alors la classe C possède cet attribut. De même, si une opération O est ajoutée dans une classe C, alors la classe C peut réaliser l'opération O. Etre confronté au résultat de l'opération qu'il réalise peut aider l'apprenant à contrôler son action. Il peut vérifier mentalement si le résultat de l'action correspond à ce qu'il souhaite obtenir ou si cela a un sens dans le langage UML. Pour les relations, nous proposons deux messages d'aide. Le premier est affiché au survol de la première classe sélectionnée (C1 dans le tableau 4.1). Comme le message de création d'une classe, il s'agit ici de rappeler le type de la relation créée et de préciser le rôle joué dans la relation par la classe sélectionnée. Par exemple pour un héritage, la première classe sélectionnée est la classe fille. Le second message est affiché au survol de la seconde classe (C2). Il reformule le résultat de l'action, c'est-à-dire la relation en cours de création. Pour un héritage, si on a choisi une classe C1 et qu'on se positionne sur une classe C2, le message sera « C1 est une sorte de C2 ».

TAB. *4.1 – Messages d'aide à la création*

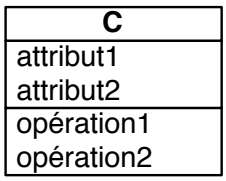
Actions en cours	**Moments d'affichage**	**Messages d'aide**
Création d'une classe C	Lorsque le curseur survole l'espace de modélisation	Création d'une classe C
Création d'une classe abstraite C	Lorsque le curseur survole l'espace de modélisation	Création d'une classe abstraite C
Création d'une classe interface C	Lorsque le curseur survole l'espace de modélisation	Création d'une classe interface C
Ajout d'un attribut A dans la classe C	Lorsque le curseur survole la classe C	Création d'un attribut A. Est-ce que C possède l'attribut A ?
Ajout d'une opération O dans la classe C	Lorsque le curseur survole la classe C	Création d'une opération O. Est-ce que C peut réaliser l'opération O ?
Ajout d'une association entre les classes C1 et C2 	Lorsque le curseur survole la classe C1	Création d'une association. Classe source : C1 ?
	Lorsque le curseur survole la classe C2	Création d'une association. C1 est associé(e) à C2 ?
Ajout d'une agrégation entre les classes C1 et C2 	Lorsque le curseur survole la classe C1	Création d'une agrégation. Classe agrégat (le tout) : C1 ?
	Lorsque le curseur survole la classe C2	Création d'une agrégation. C1 est un ensemble de C2 ?
Ajout d'une composition entre les classes C1 et C2. 	Lorsque le curseur survole la classe C1	Création d'une composition. Classe composite (le tout) : C1 ?
	Lorsque le curseur survole la classe C2	Création d'une composition. C1 est composé(e) de C2 ?

TAB. *4.1 – (suite)*

Actions en cours	Moments d'affichage	Messages d'aide
Ajout d'un héritage entre les classes C1 et C2 	Lorsque le curseur survole la classe C1	Création d'un héritage. Classe fille : C1 ?
	Lorsque le curseur survole la classe C2	Création d'un héritage. C1 est une sorte de C2 ?
Ajout d'un lien de réalisation entre la classe C1 et l'interface C2 	Lorsque le curseur survole la classe C1	Création d'un lien de réalisation. Classe source : C1 ?
	Lorsque le curseur survole l'interface C2	Création d'un lien de réalisation. C1 réalise l'interface C2 ?
Ajout d'un lien de dépendance entre les classes C1 et C2 	Lorsque le curseur survole la classe C1	Création d'un lien de dépendance. Classe source : C1 ?
	Lorsque le curseur survole la classe C2	Création d'un lien de dépendance. C1 dépend de C2 ?

4.2.3.2 Reformulation des éléments graphiques

Concernant la reformulation des éléments graphiques du diagramme de classes, nous proposons une reformulation des relations et de tous les types de classes. Pour les classes, nous choisissons d'indiquer le type de classe (classe, classe abstraite ou classe interface) ainsi que les caractéristiques de la classe, c'est-à-dire les attributs et les opérations comme le montre la figure 4.5.



C est une classe
C possède l'attribut *attribut1*
C possède l'attribut *attribut2*
C peut réaliser l'opération *opération1*
C peut réaliser l'opération *opération2*

FIG. *4.5 – Reformulation d'une classe*

En ce qui concerne les relations, nous proposons un message de reformulation qui contient le type de relation (association, agrégation, composition, héritage, réalisation, dépendance) et une reformulation de la signification de la relation. Pour cela, nous proposons d'utiliser le nom de la relation s'il existe. Ainsi pour une relation nommée R entre deux classes C1 et C2 nous obtenons le message « C1 R C2 ». Si la relation n'est pas nommée, la reformulation est construite à l'aide du sens commun de la relation comme le présente le tableau 4.2.

Dans le domaine particulier de la modélisation orientée objet, nous avons constaté que les apprenants éprouvent souvent des difficultés à lire correctement les multiplicités et inversent fréquemment leur sens de lecture. Nous proposons d'ajouter à la reformulation des associations, agrégations et compositions, une reformulation des multiplicités en utilisant les valeurs usuelles de celles-ci dans la notation UML (que nous avons décrit

dans le tableau 2.1 page 21) afin d'expliciter leur signification. La figure 4.6 présente la reformulation complète (type, nom et multiplicités) de deux relations.

TAB. *4.2 – Reformulation des relations*

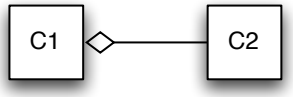
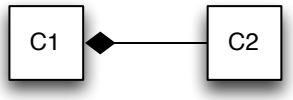
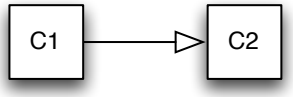
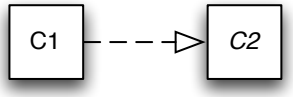
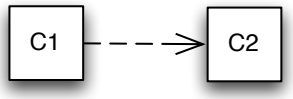
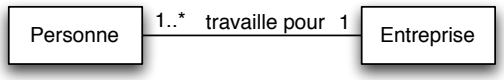
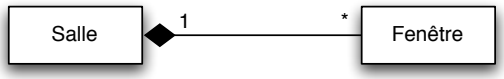
Relations	Reformulations
	Association C1 est associé(e) à C2
	Agrégation C1 est un ensemble de C2
	Composition C1 est composé(e) de C2
	Lien d'héritage C1 est une sorte de C2
	Lien de réalisation C1 réalise C2
	Lien de dépendance C1 dépend de C2
	Association Personne travaille pour Entreprise Un(e) Personne est associé(e) à un(e) unique Entreprise Un(e) Entreprise est associé(e) à au moins un(e) Personne
	Composition Salle est composé(e) de Fenêtre Un(e) Salle est associé(e) à zéro ou plusieurs Fenêtre Un(e) Fenêtre est associé(e) à un(e) unique Salle

FIG. *4.6 – Exemple de relations reformulées*

4.2.4 Rétroactions

L'objectif des rétroactions que nous proposons dans le cadre de l'apprentissage de la modélisation par diagrammes de classes UML est d'encourager l'apprenant à réfléchir à la pertinence du diagramme qu'il propose. Pour l'évaluation du diagramme de l'apprenant, nous faisons l'hypothèse que l'EIAH réifiant notre modèle dispose d'un outil de diagnostic qui compare le diagramme de l'apprenant avec un diagramme de

référence fourni par l'enseignant. Les différences repérées sont basées sur la structure des diagrammes mais ne sont pas directement exploitables pour produire des rétroactions. Pour construire les rétroactions, nous avons élaboré une taxonomie de différences qualifiées de « pédagogiques », entre le diagramme de l'apprenant et le diagramme de référence. A partir des différences pédagogiques, nous élaborons les rétroactions sous la forme de messages, selon les trois modalités définies : indiquer, questionner et proposer. L'évaluation du diagramme se déroulant une fois que l'activité est jugée terminée par l'apprenant, nous avons modifié la méthode de modélisation afin d'intégrer l'évaluation du diagramme de l'apprenant et les rétroactions pédagogiques.

Nous présentons dans le paragraphe suivant l'extension de la méthode pour prendre en compte les rétroactions qui seront produites après la phase de modélisation. Ensuite, nous détaillons la taxonomie que nous avons élaborée de différences dites pédagogiques ainsi que le principe d'élaboration des rétroactions. Enfin, nous donnons un exemple du contenu et de la forme des rétroactions : à partir du diagramme de l'apprenant nous donnons la liste des différences pédagogiques repérées et nous décrivons les rétroactions correspondantes.

4.2.4.1 Intégration des rétroactions dans les activités de modélisation : extension de la méthode de modélisation

L'évaluation du diagramme de classes de l'apprenant nécessite que le modèle de l'apprenant soit complet ou quasi-complet. Sur un modèle incomplet, les éléments non encore modélisés ne peuvent pas être distingués des éléments omis, et le système les considérerait comme des omissions. L'intégration de l'évaluation du modèle et des rétroactions nous a donc conduit à modifier la scénarisation globale des activités : nous avons choisi d'intégrer la phase d'analyse du diagramme à l'issue de l'étape de relecture. L'évaluation du diagramme s'effectuant une fois la tâche réalisée, c'est principalement la fonction de vérification qui sera visée. Nous proposons une extension de la méthode de modélisation en trois étapes afin d'intégrer une étape d'affichage des messages de rétroaction dans l'EIAH (cf. figure 4.7).

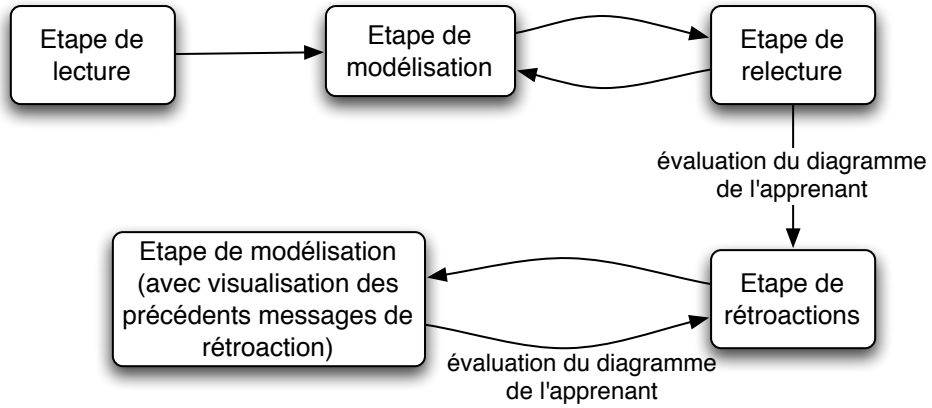


FIG. *4.7 – Extension de la méthode de modélisation pour l'intégration des rétroactions*

A l'issue de l'étape de relecture, nous construisons les messages de rétroaction à destination de l'apprenant. Si celui-ci le souhaite, il peut accéder à une nouvelle étape de modélisation afin de modifier son diagramme en fonction des rétroactions reçues. Pour cette cinquième étape nous proposons de conserver à l'écran les messages de rétroaction élaborés à partir de la dernière évaluation du diagramme de l'apprenant. Il est possible de

revenir de la cinquième à la quatrième étape afin de générer une nouvelle évaluation du diagramme et de nouveaux messages de rétroaction.

4.2.4.2 Taxonomie de différences « pédagogiques »

L'élaboration des rétroactions est basée sur une taxonomie de différences que nous appelons différences pédagogiques entre le diagramme de l'apprenant et le diagramme de référence. Nous avons construit la taxonomie de différences pédagogiques en appliquant la méthode suivante : dans un premier temps nous avons recueilli les diagrammes produits par les apprenants lors d'expérimentations réalisées à l'automne 2006 et en mars 2007. Nous avons sélectionné deux exercices pour lesquels nous avons obtenu un grand nombre de diagrammes produits par les apprenants (26 et 30 diagrammes). Nous considérons, grâce à une analyse *a priori*, que les deux exercices choisis sont représentatifs de la base d'exercices que nous avons utilisée, du point de vue de leur difficulté et de la variété des éléments contenus dans le diagramme de référence fourni par l'enseignant. Pour chaque exercice, nous avons comparé la solution de l'apprenant avec le diagramme de référence afin de noter les différences sur lesquelles une rétroaction était possible. Nous avons ainsi obtenu une liste exhaustive de différences. A partir de cette liste, nous avons établi la taxonomie de différences « simples » en huit catégories :

- l'*omission* par l'apprenant d'un élément du diagramme de référence.
- l'*ajout* par l'apprenant d'un élément qui ne figure pas dans le diagramme de référence.
- le *transfert* d'un élément vers une autre partie du diagramme. Par exemple, une relation entre deux classes C1 et C2 dans le diagramme de référence est déplacée par l'apprenant entre les classes C1 et C3.
- le *dédoublement* d'un élément : un élément du diagramme de référence est représenté dans le diagramme de l'apprenant par plusieurs éléments du même type.
- la *fusion* d'éléments : plusieurs éléments d'un même type sont représentés dans le diagramme de l'apprenant par un seul élément.
- la *représentation erronée* d'un élément : 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.
- l'*inversion* du sens d'une relation orientée (comme l'héritage par exemple) 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.

Chaque catégorie regroupe un ensemble de différences, selon la nature des éléments du diagramme concernés par la différence. Par exemple, dans la catégorie *ajout*, on retrouve les différences suivantes :

- Ajout d'une classe (d'une classe abstraite ou d'une interface) ;
- Ajout d'un attribut ;
- Ajout d'une opération ;
- Ajout d'une relation (association, agrégation, composition, héritage, réalisation ou dépendance).

Dans les diagrammes des apprenants, nous avons constaté que certaines différences s'accompagnent souvent d'autres différences simples. Par exemple, la différence d'Omission de classe peut entraîner les différences d'Omission d'une relation ou de

Transfert d'une relation, car si l'apprenant ne représente pas une classe, alors il a probablement omis ou transféré les relations liées à cette 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. Nous avons donc défini un ensemble de différences « composées » de manière à les appréhender globalement. Une différence composée est constituée d'une différence principale et d'un ensemble de différences associées, comme le montre le tableau 4.3.

TAB. *4.3 – Exemples de différences composées*

Différences composées	**Différences principales**	**Différences associées**
Omission d'une classe et des éléments liés à cette classe	Omission d'une classe	Omission d'une relation liée à cette classe, omission d'un attribut de cette classe, dédoublement et/ou transfert d'une relation liée à cette classe, etc.
Dédoublement et transfert d'une relation	Dédoublement d'une relation	Transfert de cette relation, erreur sur le type de cette relation, inversion du sens de cette relation, différence de multiplicités.
Représentation erronée d'une relation et inversion de sens	Inversion du sens d'une relation	Type erroné de relation

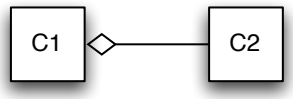
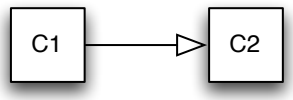
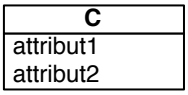
4.2.4.3 Elaboration des rétroactions

Le processus d'élaboration des rétroactions associées aux différences pédagogiques a été réalisé en collaboration avec Thierry Lemeunier, enseignant référent du projet. Pour chaque différence pédagogique simple et composée, nous avons retenu une ou plusieurs modalités d'intervention parmi l'indication, la question et la proposition (cf. paragraphe 4.1.4, page 67) que nous jugeons pertinentes dans ce cas. Pour chaque modalité, nous avons élaboré un message textuel. Le tableau 4.4 présente quelques exemples de rétroactions, associées à des différences simples.

La différence simple d'ajout d'une classe implique que celle-ci est isolée car sinon nous aurions également une différence d'ajout de relation ou de transfert de relation. Dans ce cas, nous proposons une rétroaction en deux modalités. L'indication présente le problème à l'apprenant : la classe n'est pas reliée au diagramme. La question est alors une question ouverte que l'apprenant doit se poser pour vérifier son diagramme et déterminer si la classe ajoutée est utile ou non dans le diagramme qu'il modélise. Nous n'énonçons aucune proposition car un ajout peut signifier que l'apprenant construit une variante du diagramme de référence.

Dans le cas d'une omission de relation, nous suggérons une rétroaction en trois modalités. L'indication informe l'apprenant qu'une relation manque dans son diagramme, et la question l'invite à réfléchir sur son diagramme, en faisant référence aux concepts qu'il a déjà modélisés. Enfin, la proposition suggère la relation omise, en utilisant la

TAB. *4.4 – Exemples de rétroactions associées à des différences simples*

Différences	**Rétroactions**
Ajout d'une classe	**Indiquer** : La classe C n'est pas reliée au diagramme **Questionner** : La classe C est-elle utile ?
Omission d'une relation : cas de l'héritage	**Indiquer** : Il manque une relation entre les classes C1 et C2 **Questionner** : Quelle(s) relation(s) existent entre C1 et C2 ? **Proposer** : C1 est une sorte de C2
Représentation erronée d'une relation : cas d'une agrégation au lieu d'une composition 	**Indiquer** : Tu dis que C1 est un ensemble de C2 **Questionner** : Est-ce que la relation entre C1 et C2 n'est pas plus forte ? Est-ce qu'un objet C2 peut être partageable entre plusieurs objets C1 ?
Inversion de sens d'une relation : cas de l'héritage 	**Indiquer** : Tu dis que C1 est une sorte de C2 **Questionner** : Est-ce que C1 est une sorte de C2 ? **Proposer** : Je dirais plutôt que c'est C2 qui est une sorte de C1
Dédoublement d'attributs (ici de deux attributs) 	**Questionner** : Les attributs attribut1 et attribut2 de la classe C représentent-ils la même propriété ? **Proposer** : Tu dois n'en garder qu'un seul

reformulation usuelle de la relation.

Pour une différence simple de type erroné de relation et dans le cas d'une agrégation au lieu d'une composition, la rétroaction se fait en deux modalités. L'indication reformule la relation telle que l'apprenant l'a représentée. Nous avons fait le choix d'utiliser la reformulation du type de la relation plutôt que d'utiliser le nom de la relation, pour insister sur la nature de la relation. Dans le cas d'une agrégation au lieu d'une composition, la question interroge l'apprenant sur la nature de la relation. Elle utilise alors ce qui fait la différence entre ces deux relations, c'est-à-dire le fait que les composants ne soient pas partageables dans le cas de la composition comme nous l'avons expliqué dans le chapitre 2, paragraphe 2.2.3, page 19. La composition étant une agrégation plus forte, nous ne suggérons pas la modification à apporter car la nuance entre les deux peut être délicate dans de nombreux cas. Dans d'autres cas de type erroné de relation, nous suggérons une modification, comme dans le cas d'un héritage au lieu d'une association par exemple (cf. annexe A, page 135).

La différence d'inversion de sens d'une relation orientée peut provenir d'une mauvaise compréhension de la sémantique des éléments du langage UML : nous utilisons donc la reformulation des relations pour mettre en évidence le problème repéré. Comme dans le cas du type erroné de relation, même si le nom de la relation est précisé, nous utilisons

la reformulation du type de relation pour attirer l'attention de l'apprenant sur le type de la relation et sur son orientation. L'indication reformule la signification de la relation de l'apprenant et la question se présente comme une demande de confirmation sur la représentation de la relation. Comme la présence d'une erreur est très probable dans ce cas, nous formulons une proposition plus directive, qui suggère la correction à apporter au diagramme comme l'enseignant pourrait le faire dans ce cas là : « Je dirais plutôt que . . . ».

Lorsque l'apprenant représente plusieurs attributs qui ont le même sens, nous repérons une différence de dédoublement d'attributs. La rétroaction interroge l'apprenant sur les attributs concernés pour que celui-ci détermine s'ils représentent la même propriété. La proposition suggère alors de n'en garder qu'un seul.

Pour l'élaboration des rétroactions liées à des différences composées, nous pensons qu'il est préférable de cibler la rétroaction sur le ou les éléments ayant le plus d'importance. Ainsi, dans certains cas, la rétroaction d'une différence composée sera celle de la différence principale qui la compose. Dans d'autres cas, la différence composée mènera à une rétroaction spécifique. Le tableau 4.5 présente des exemples de rétroactions associées à des différences composées.

TAB. *4.5 – Exemples de rétroactions associées à des différences composées*

Différences composées	**Rétroactions**
Ajout d'une classe, dédoublement, transfert ou ajout de relations liées à cette classe	**Indiquer** : Tu as représenté une classe C **Questionner** : La classe C est-elle nécessaire ? La classe C représente-elle un concept important du domaine ?
Omission d'une classe et des relations liées à cette classe	**Indiquer** : Le diagramme est incomplet : un concept important n'est pas représenté **Questionner** : As-tu représenté tous les concepts dans le diagramme ?
Représentation erronée d'une relation et multiplicités erronées : cas d'une agrégation au lieu d'une composition	**Indiquer** : Tu dis que C1 est un ensemble de C2 **Questionner** : Est-ce que la relation entre C1 et C2 n'est pas plus forte ? Est-ce qu'un objet C2 peut être partageable entre plusieurs objets C1 ?
Représentation erronée d'une relation et **inversion de sens** : cas d'une agrégation au lieu d'une composition,	**Indiquer** : Tu dis que C1 est un ensemble de C2 **Questionner** : Est-ce que C1 est un ensemble de C2 ? **Proposer** : Je dirais plutôt que c'est C2 qui est un ensemble de C1

On retrouve le cas de l'ajout d'une classe, mais contrairement à la différence simple, des relations lient cette classe au reste du diagramme. La rétroaction est donc différente. L'indication précise l'élément concerné par la rétroaction : la classe ajoutée. La question interroge l'apprenant sur l'importance de la classe ajoutée.

Pour une différence d'omission d'une classe et des relations liées à cette classe, nous établissons une rétroaction pédagogique en deux modalités, qui se focalise sur la différence principale d'omission de classe. L'indication précise le problème repéré : un concept est

manquant dans le diagramme et la question confronte l'apprenant à son diagramme pour qu'il évalue puis corrige son diagramme, si cela lui semble nécessaire. Nous avons choisi de ne pas ajouter de proposition car les concepts à modéliser ne sont pas toujours explicitement indiqués dans l'énoncé.

Lorsque nous repérons une différence de représentation d'une relation (utilisation d'un autre type de relation) et que nous repérons également des multiplicités différentes, nous utilisons la même rétroaction que pour la différence simple de type de relation erroné. En effet il nous semble plus important de concentrer l'attention de l'apprenant sur la vérification du type de la relation, l'indication des multiplicités intervenant au second plan pour clarifier le diagramme.

La différence de mauvaise représentation, dans le cas d'une agrégation au lieu d'une composition, peut s'accompagner de la différence d'inversion du sens de la relation. La rétroaction associée à cette différence composée est focalisée sur la différence d'inversion du sens de la relation, que nous jugeons plus importante à corriger. Les modalités sont alors les mêmes que dans le cas de la différence simple d'inversion de sens.

4.2.4.4 Exemple

Nous présentons ici un exemple concret afin d'illustrer les rétroactions élaborées à partir du diagramme d'un apprenant. La figure 4.8 montre le diagramme de l'apprenant et le diagramme de référence auquel nous le comparons.

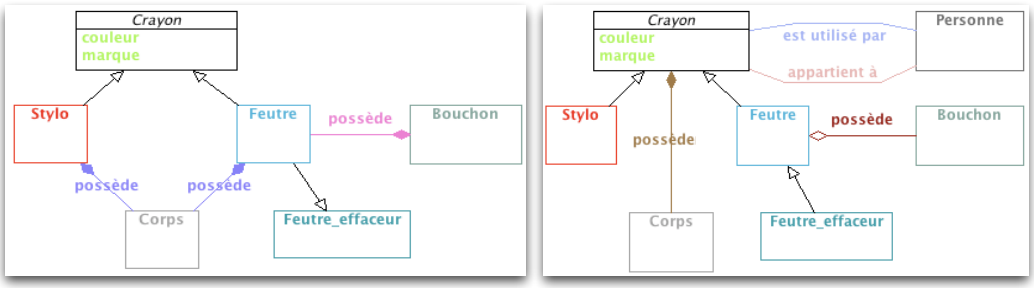


FIG. *4.8 – Diagramme de l'étudiant (à gauche) et diagramme de référence (à droite)*

Cette comparaison nous donne neuf différences pédagogiques simples. Trois ensembles de différences simples sont regroupés en différences composées (A, B et C), la dernière différence simple (D) est isolée (cf. tableau 4.6). A chaque différence pédagogique simple et composée correspond un message de rétroaction comme nous l'avons expliqué dans le paragraphe précedent.

TAB. *4.6 – Différences pédagogiques simples et composées de l'exemple*

Différences simples	**Différences composées**
Omission de la classe 'Personne'	(A) Omission d'une classe et des relations liées à cette classe
Omission de la relation 'appartient à' entre 'Personne' et 'Crayon'	
Omission de la relation 'est utilisé par' entre 'Personne' et 'Crayon'	
Dédoublement de la relation 'possède' entre 'Corps' et 'Feutre' et entre 'Corps' et 'Stylo'	(B) Dédoublement et transfert d'une relation
Transfert de la relation 'possède' entre 'Corps' et 'Crayon' vers 'Corps' et 'Stylo'	
Transfert de la relation 'possède' entre 'Corps' et 'Crayon' vers 'Corps' et 'Feutre'	
Inversion du sens de la relation entre 'Feutre' et 'Bouchon'	(C) Représentation erronée d'une relation et inversion de sens
Composition au lieu de Agrégation entre 'Feutre' et 'Bouchon'	
(D) Inversion du sens de l'héritage entre 'Feutre' et 'Feutre_effaceur'	

Dans notre exemple, l'apprenant a omis la classe 'Personne', ainsi que les relations 'est utilisé par' et 'appartient à' entre les classes 'Crayon' et 'Personne'. Ces différences simples forment une différence pédagogique composée, qui exprime l'omission d'une classe et des éléments liés à cette classe (différence A). A partir de cette différence, nous établissons une rétroaction pédagogique en deux modalités, qui se focalise sur la différence principale d'omission de classe (cf. figure 4.9).

```
Indiquer : Le diagramme est incomplet : un concept important n'a
pas été représenté
Questionner : As-tu représenté tous les concepts dans le
diagramme ?
```

FIG. *4.9 – Rétroaction associée à la différence composée (A)*

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 des différences pédagogiques. Dans ce cas précis une rétroaction spécifique est construite. Elle est constituée de trois modalités qui font référence à la fois au dédoublement et au transfert des éléments et qui s'appuient sur le diagramme de l'apprenant (cf. figure 4.10). L'indication reformule les deux relations concernées par le message. Nous avons élaboré trois questions. La première interroge l'apprenant sur ce que représentent les relations. Les deux autres questions font référence aux propriétés de l'héritage et guident l'apprenant vers les éléments à considérer pour

évaluer son diagramme c'est-à-dire le fait que les deux classes vers lesquelles les relations ont été dédoublées ont une classe mère. La proposition suggère d'utiliser le processus de généralisation pour regrouper les relations, en utilisant 'Crayon', la classe mère de 'Stylo' et 'Feutre'.

Indiquer : Tu dis que **Stylo** est composé de **Corps** et que **Feutre** est composé de **Corps**
Questionner : Les relations **possède** entre **Stylo** et **Corps** et **possède** entre **Feutre** et **Corps** représentent-elles la même relation ? Dans un héritage, qu'est ce qui est hérité par les sous-classes ? Dans un héritage, les relations de la classe mère sont-elles héritées par les sous-classes ?
Proposer : Tu dois regrouper les relations en une seule, entre les classes **Crayon** et **Corps**

FIG. *4.10 – Rétroaction associée à la différence composée (B)*

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 simples correspondent à une différence pédagogique composée (C), formée de la différence principale d'inversion de sens et de la différence associée d'erreur de type. La rétroaction associée à cette différence composée, présentée figure 4.11, est focalisée sur la différence principale d'inversion du sens de la relation, comme nous l'avons expliqué précédemment.

Indiquer : Tu dis que **Bouchon** est composé de **Feutre**
Questionner : Est-ce que **Bouchon** est composé de **Feutre** ?
Proposer : Je dirais plutôt que **Feutre** est composé de **Bouchon**

FIG. *4.11 – Rétroaction associée à la différence composée (C)*

Enfin, l'apprenant a inversé le sens de l'héritage entre 'Feutre effaceur' et 'Feutre'. Cette différence pédagogique simple (D) conduit à une rétroaction isolée, présentée figure 4.12, dont les modalités sont semblables à celle de la différence (C).

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**

FIG. *4.12 – Rétroaction associée à la différence isolée (D)*

Au total, quatre rétroactions sont produites pour ce diagramme, dont trois concernent des différences composées et une correspond à une différence simple. Chaque rétroaction se décline en deux ou trois modalités selon le cas, et donne lieu à des messages plus ou moins directifs. Ces messages sont présentés lors de l'étape de rétroactions.

4.3 Conclusion

Dans ce chapitre, nous avons exposé notre principale contribution : un modèle d'interaction pour un EIAH d'apprentissage de la modélisation informatique. Nous avons décrit notre modèle selon deux niveaux d'études. En premier lieu, nous avons décrit les

éléments génériques du modèle, applicables à différents domaines de la modélisation informatique. Ensuite, nous avons étudié plus précisément l'application du modèle à la modélisation orientée objet, et en particulier à la modélisation par diagrammes de classes dans le langage UML. De cette application, nous pouvons tirer des limites à la généricité de notre modèle. En effet, la formulation des messages d'aide est dépendante de la sémantique des éléments du domaine. De même, pour être pertinentes, les rétroactions doivent prendre en compte le travail de l'apprenant et doivent être adaptées au domaine de modélisation.

Nous présentons dans le chapitre suivant la mise en œuvre de notre modèle dans DIAGRAM, un environnement informatique pour l'apprentissage de la modélisation orientée objet, développé dans le cadre du projet « Interaction et connaissances dans les EIAH pour la modélisation ». Il s'agit de rendre le modèle opérationnel et ainsi de montrer sa faisabilité.

CHAPITRE 5

Mise en œuvre du modèle d'interaction et réalisation informatique

Sommaire

Résumé

Ce chapitre présente la mise en œuvre des éléments de notre modèle d'interaction dans DIAGRAM, un environnement informatique pour l'apprentissage humain développé dans le cadre du projet « Interaction et connaissances dans les EIAH pour la modélisation ». Dans un premier temps, nous présentons comment chaque élément de notre modèle est implémenté dans DIAGRAM. Dans un second temps nous abordons les principes

de réalisation informatique de DIAGRAM. Nous présentons l'architecture générale de DIAGRAM et détaillons certains modules en précisant nos contributions en terme de réalisation informatique.

5.1 Mise en œuvre du modèle d'interaction dans l'EIAH DIAGRAM

Dans cette section, nous présentons la mise en oeuvre des quatre éléments de notre modèle d'interaction dans DIAGRAM. En premier lieu, nous décrivons comment l'interface nous permet d'intégrer les modes d'interaction. Ensuite, nous détaillons les implémentations des aides métacognitives de création et de reformulation des éléments. Dans un troisième temps nous présentons la mise en œuvre des rétroactions dans DIAGRAM en lien avec la méthode de diagnostic utilisée dans le système. Enfin, nous illustrons l'utilisation de DIAGRAM par un exemple complet du déroulement d'une activité.

5.1.1 Mise en œuvre des modes d'interaction

Ce paragraphe présente l'intégration des modes d'interaction dans DIAGRAM. Pour cela, nous présentons l'interface de DIAGRAM et les outils de modélisation. Les activités structurées en étapes seront présentées dans le paragraphe 5.1.4.

5.1.1.1 Interface de DIAGRAM

Deux espaces de l'interface de DIAGRAM permettent de mettre en œuvre les outils de modélisation. Le premier espace intègre l'énoncé et le second contient le diagramme de classes.

Espace intégrant l'énoncé

L'intégration de l'énoncé à l'interface se fait dans DIAGRAM à l'aide d'un espace dédié, appelé « éditeur textuel ». Il est constitué de deux éléments : une barre d'outils et un espace dans lequel l'énoncé sera affiché, comme le montre la figure 5.1. La barre d'outils contient les boutons permettant d'agir sur le texte. Deux boutons permettent de modifier la police de caractères de l'énoncé, le troisième permet de souligner le texte. Enfin, la barre d'outils contient un bouton permettant de supprimer la couleur d'une expression.

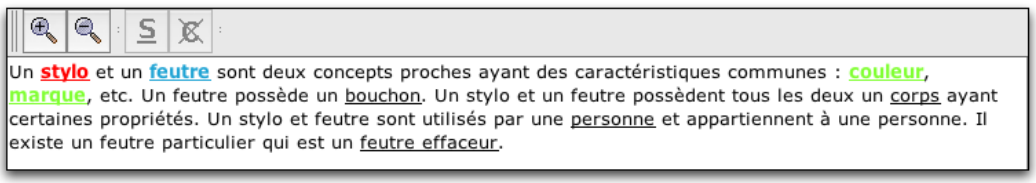


FIG. *5.1 – Interface de l'espace contenant l'énoncé*

Espace de modélisation graphique

L'espace de modélisation graphique, appelé « éditeur graphique » est destiné à contenir le modèle réalisé par l'apprenant. Il est constitué d'une barre d'outils et de l'espace de création lui-même (figure 5.2).

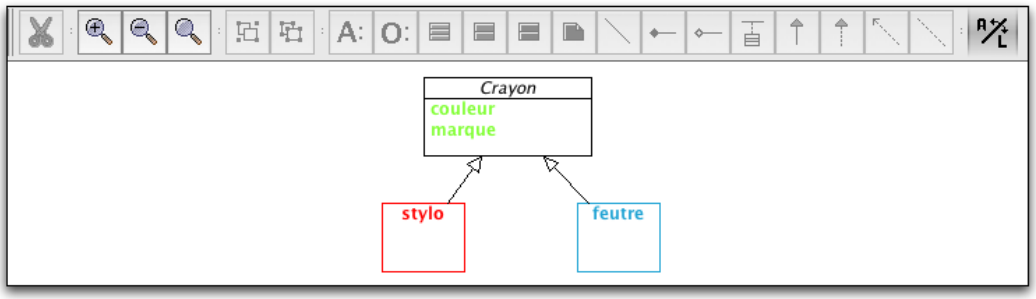


FIG. *5.2 – Interface de l'éditeur graphique*

La barre d'outils de l'éditeur graphique permet d'une part d'accéder directement aux fonctions standards d'édition telles que le zoom avant, le zoom arrière, la suppression d'éléments graphiques, etc. D'autre part, elle contient un bouton par élément du langage UML qu'il est possible de créer dans DIAGRAM. Ces éléments sont ceux que nous traitons dans notre modèle : attribut, opération, classe, classe abstraite, interface, association, agrégation, héritage, réalisation, dépendance.

L'éditeur graphique fonctionne selon les deux modes définis dans le chapitre 4, paragraphe 4.1.2.3, page 65. Contrairement au mode « libre », le mode « assisté » exige que l'apprenant utilise une expression de l'énoncé pour créer un élément du diagramme. La barre d'outils contient un bouton permettant de basculer d'un mode de fonctionnement à l'autre. Nous avons choisi le mode assisté comme mode de fonctionnement par défaut de DIAGRAM afin d'inciter les apprenants à utiliser l'énoncé pour s'appuyer sur le texte lors de la modélisation du diagramme. L'apprenant peut choisir de passer en mode libre pour créer un élément, mais nous avons fait le choix de remettre le mode assisté après toute création en mode libre.

5.1.1.2 Outils de modélisation

Ce paragraphe décrit la réification dans DIAGRAM des outils de soulignage, coloriage, rattachement et passage de la souris et précise leur fonctionnement.

Outil de soulignage

L'outil de soulignage est intégré dans DIAGRAM à l'aide d'un bouton dans la barre d'outils. Il est accessible dès lors que l'éditeur de texte est disponible. Son fonctionnement est similaire à celui d'un éditeur de texte classique. Pour souligner une expression, il faut la sélectionner à l'aide de la souris et cliquer sur le bouton de soulignage. Nous avons ajouté une facilité de sélection des expressions soulignées : on peut les sélectionner en un seul clic. Pour désouligner une expression, il suffit de la sélectionner et de cliquer de nouveau sur le bouton de soulignage. Pour des raisons de cohérence de l'interface et de cohérence des données, nous avons empêché le chevauchement d'expression soulignées, c'est-à-dire qu'il n'est pas possible de souligner ou désouligner une expression qui contient ou qui chevauche une expression soulignée. En effet, les expressions soulignées représentent les éléments jugés importants par l'apprenant pour la modélisation et doivent être manipulables comme des entités d'un seul bloc.

Outil de coloriage

L'outil de coloriage permet de créer un élément graphique à partir d'une expression du texte. Pour y parvenir, il faut que le mode de fonctionnement en cours soit le mode assisté.

Pour créer un élément en mode assisté, il faut en premier lieu sélectionner une expression de l'énoncé, puis choisir l'élément. Ensuite le fonctionnement dépend de l'élément en question :

- Pour une classe, on clique dans l'espace de modélisation. La classe est créée, avec une nouvelle couleur. Le nom de la classe est celui de l'expression, et l'expression et la classe sont coloriées dans la même couleur.
- Pour une relation, il faut cliquer sans relâcher sur la première classe, et relâcher le bouton sur la seconde classe. La relation est nommée par défaut avec le nom de l'expression. L'expression et la relation sont affichées dans la même couleur.
- Pour la création d'un attribut ou d'une opération, il faut cliquer sur la classe dans laquelle on souhaite ajouter l'élément. Si la classe a été créée en mode assisté, la couleur de la classe est appliquée à la propriété créée et à l'expression textuelle. Sinon, l'attribut ou l'opération créé ainsi que l'expression de l'énoncé prennent une nouvelle couleur.

L'utilisation de l'outil de coloriage génère la création d'un lien entre les deux éléments, matérialisé à l'interface par la colorisation en une même couleur de l'expression et de l'élément graphique. La figure 5.3 montre quatre exemples de liens. Dans cet exemple, les classes 'Stylo', 'Feutre' et 'Personne' ont été créées avec les mots correspondants alors que la classe 'Crayon' a été créée en mode libre. Pour les attributs 'couleur' et 'marque', la même couleur a été utilisée car ce sont deux éléments communs à une même classe. Enfin, l'association 'est utilisé par' a été créée avec l'expression 'sont utilisés par'. Cet exemple illustre le fait que le lien de couleur est maintenu même si les éléments graphiques sont renommés.

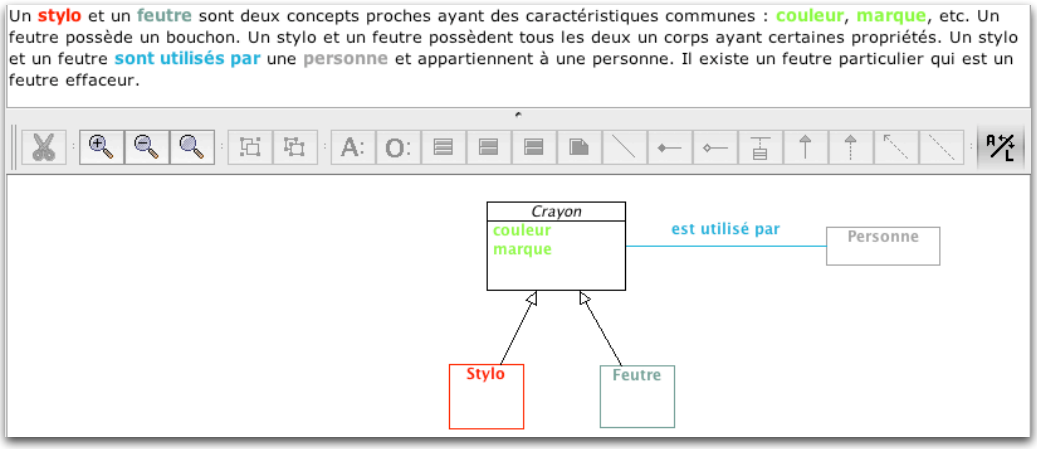


FIG. *5.3 – Outil de coloriage*

Le lien créé avec l'outil de coloriage est un lien fort : si l'expression textuelle est « décoloriée » – c'est-à-dire que la couleur de l'expression est supprimée et que celle-ci redevient noire – alors l'élément graphique créé à partir de cette expression est supprimé. Réciproquement si un élément graphique est supprimé, alors les éventuels liens vers l'énoncé sont supprimés.

La procédure de création d'un élément graphique en mode libre est la même qu'en

mode assisté sauf que l'apprenant n'a pas à sélectionner une expression. Dans ce cas, l'élément graphique sera affiché en noir, comme la classe *Crayon* de la figure 5.3. Un élément créé en mode libre peut par la suite être rattaché à une expression textuelle grâce à l'outil de rattachement.

Comme pour les expressions soulignées, nous avons ajouté une facilité de sélection en un seul clic des expressions coloriées. Nous avons également ajouté des contraintes de coloriage : il n'est pas possible d'utiliser une expression qui chevauche une expression déjà en couleur pour l'utilisation de l'outil de coloriage. Cette contrainte a été ajoutée pour faciliter la cohérence visuelle à l'écran des couleurs de l'éditeur.

Outil de rattachement

L'outil de rattachement permet de lier une expression à un élément du graphique ou une expression à une autre pour exprimer le lien qui existe entre elles. Nous avons défini trois cas de rattachement :

- le rattachement d'une expression textuelle (A) qui n'est pas en couleur à une expression textuelle (B), coloriée. L'expression (A) prend alors la couleur de l'expression (B). Si l'expression (B) est décoloriée, l'élément graphique lié ainsi que l'expression (A) sont supprimés automatiquement.
- le rattachement d'une expression textuelle à un élément du graphique déjà colorié : la couleur de l'élément sera utilisée pour colorier l'expression.
- le rattachement d'une expression à un élément graphique créé en mode libre. Dans ce cas, une nouvelle couleur est utilisée pour colorier les deux éléments.

Pour utiliser l'outil de rattachement, il faut sélectionner une expression et cliquer avec le bouton droit sur un élément. Cet élément est soit une expression pour le premier cas de rattachement, soit une classe pour un rattachement à la classe ou à une de ses propriétés, soit une relation pour un rattachement à la relation, à une multiplicité ou à un rôle défini pour cette relation. Un menu contextuel est alors affiché et l'apprenant doit valider le choix de rattachement ou peut annuler l'action, comme comme le montre la figure 5.4. Les expressions rattachées sont gérées comme les expressions coloriées (facilité de sélection, contraintes d'utilisation, etc.)

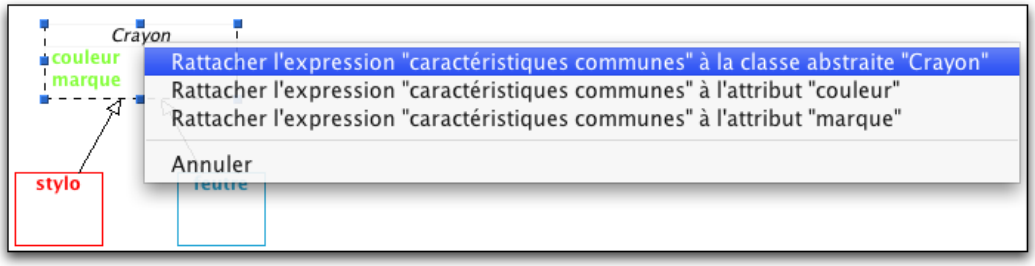


FIG. *5.4 – Menu contextuel de l'outil de rattachement*

Outil de passage de la souris

L'outil de passage de la souris est utilisé lors de de la première étape des scénarios de « Diagramme à compléter » et « Diagramme à corriger » ainsi que dans la troisième étape de chaque scénario. Au commencement de l'étape, l'énoncé est affiché en noir et les éléments graphiques, sauf ceux créés en mode libre, sont masqués, comme le montre l'image de

gauche de la figure 5.5. Pour les relations créées en mode libre, le contour des classes auxquelles elles sont liées apparaît en noir pour éviter que les relations soient affichées « dans le vide ». Au fur et à mesure du passage de la souris sur le texte, les éléments textuels coloriés et les éléments graphiques correspondants reprennent leurs couleurs, comme le montre l'image de droite de la figure 5.5.

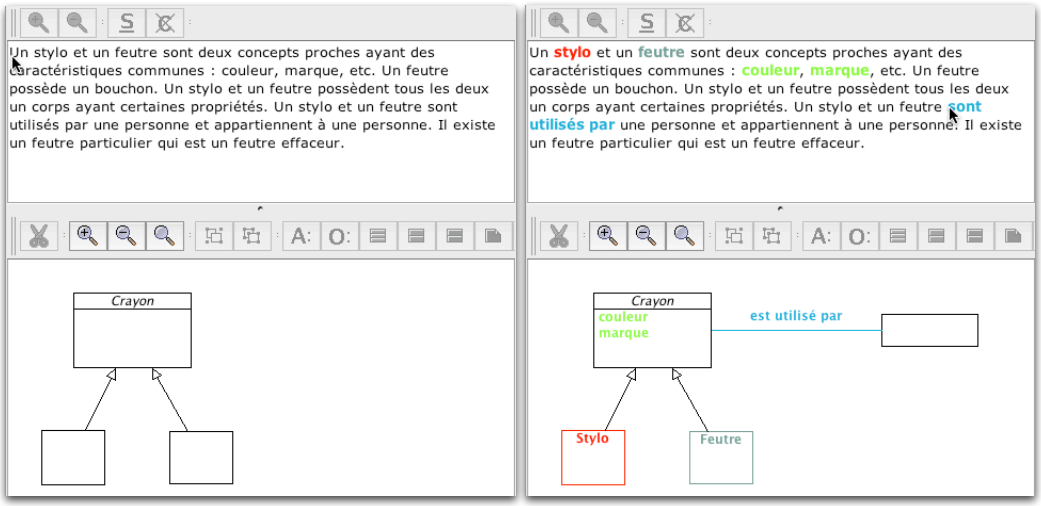


FIG. *5.5 – Outil de passage de la souris. Interface au commencement de l'étape (à gauche) et au cours de l'utilisation de l'outil de passage de la souris (à droite).*

Lorsqu'un attribut ou une opération reprend sa couleur avant la classe qui le contient, les contours de celle-ci réapparaissent en noir. De même, si une multiplicité reprend sa couleur avant la relation à laquelle elle se rapporte, cette relation apparaît en noir dans le diagramme. Cela permet d'éviter que des éléments du diagramme soient affichés « dans le vide ».

5.1.2 Aides métacognitives au cours de la modélisation

Ce paragraphe décrit l'implémentation des aides à la création et les aides de reformulation des éléments. Il donne également des exemples de leur utilisation. Nous décrivons aussi une aide supplémentaire de reformulation, permise par le fonctionnement de DIAGRAM.

5.1.2.1 Aide à la création d'éléments graphiques

L'aide à la création d'un élément graphique apparaît sous la forme d'une infobulle affichée en fonction de l'action en cours et de la position du curseur dans l'éditeur graphique. Nous distinguons trois cas : la création d'une classe, d'une relation, ou d'une propriété d'une classe. Lors de la création d'une classe, l'infobulle est affichée lorsque le curseur se situe dans l'éditeur graphique, mais sans pointer un élément existant. Comme nous l'avons défini dans le chapitre 4, paragraphe 4.2.3.1 page 72, le message rappelle simplement l'action en cours c'est-à-dire le type et éventuellement le nom de la future classe créée (cf. figure 5.6).

Concernant la création d'une relation entre deux classes, nous avons implémenté dans DIAGRAM l'affichage du premier message défini dans le chapitre 4, paragraphe 4.2.3.1, page 72, c'est-à-dire le message affiché lorsque le curseur pointe la première classe afin que

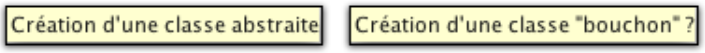


FIG. *5.6 – Exemple de messages d'aide à la création d'une classe*

l'apprenant identifie la classe source de la relation. Par exemple, la figure 5.7 nous montre les infobulles affichées pour une création d'un lien d'héritage et d'une composition.

Pour des raisons techniques, nous n'avons pas implémenté le second message (s'affichant lorsque le curseur pointe la seconde classe).

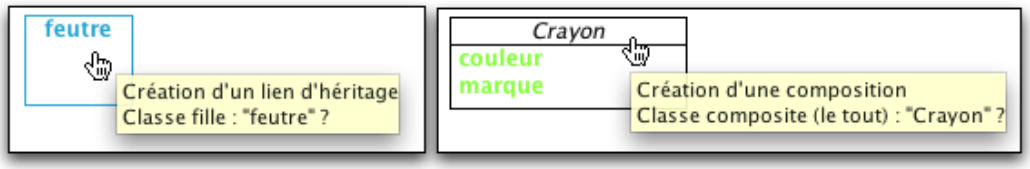


FIG. *5.7 – Exemple de messages d'aide à la création d'une relation*

Lors de la création d'une opération ou d'un attribut d'une classe, l'infobulle est affichée lorsque le curseur pointe une classe. La figure 5.8 montre l'exemple d'un message affiché pour la création d'un attribut *couleur*, et lorsque le curseur pointe la classe *Crayon*. Comme nous l'avons défini dans le chapitre 4, paragraphe 4.2.3.1, page 72, le message se compose de deux lignes. La première rappelle l'action en cours et la seconde reformule le résultat de l'action en utilisant le nom de la classe.

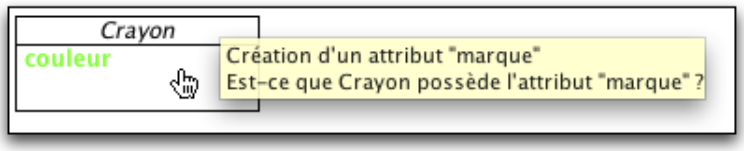


FIG. *5.8 – Exemple d'un message d'aide à la création d'un attribut*

5.1.2.2 Reformulation des éléments graphiques

Nous avons implémenté les reformulations locales des éléments sous forme d'infobulles automatiquement affichées lorsque le curseur survole une classe ou une relation. Lorsque le curseur se situe sur une classe, une classe abstraite ou une classe interface, le message de reformulation donne le nom et le type de la classe. Il précise ensuite les attributs que possède la classe et les opérations qu'elle peut réaliser (figure 5.9).

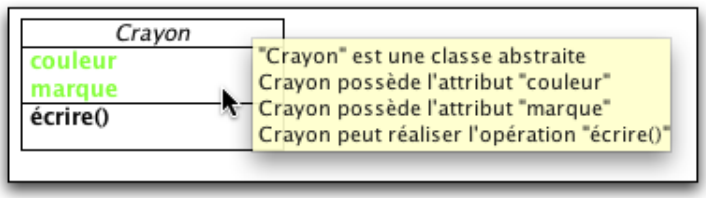


FIG. *5.9 – Exemple de reformulation d'une classe*

La figure 5.10 donne quatre exemples de reformulation de relations. Le lien d'héritage est toujours reformulé en utilisant l'expression « est une sorte de ». Les relations d'association et d'agrégation sont, dans cet exemple, reformulées en utilisant le nom de la relation. Pour ces relations, le message inclut la reformulation des multiplicités. Dans l'exemple, la relation de composition n'est pas nommée : elle est reformulée en utilisant l'expression « est composé(e) de ». Lorsque les multiplicités ne sont pas renseignées, nous avons fait le choix d'afficher tout de même les infobulles correspondantes afin d'interpeller l'apprenant sur le fait que la multiplicité n'est pas indiquée.

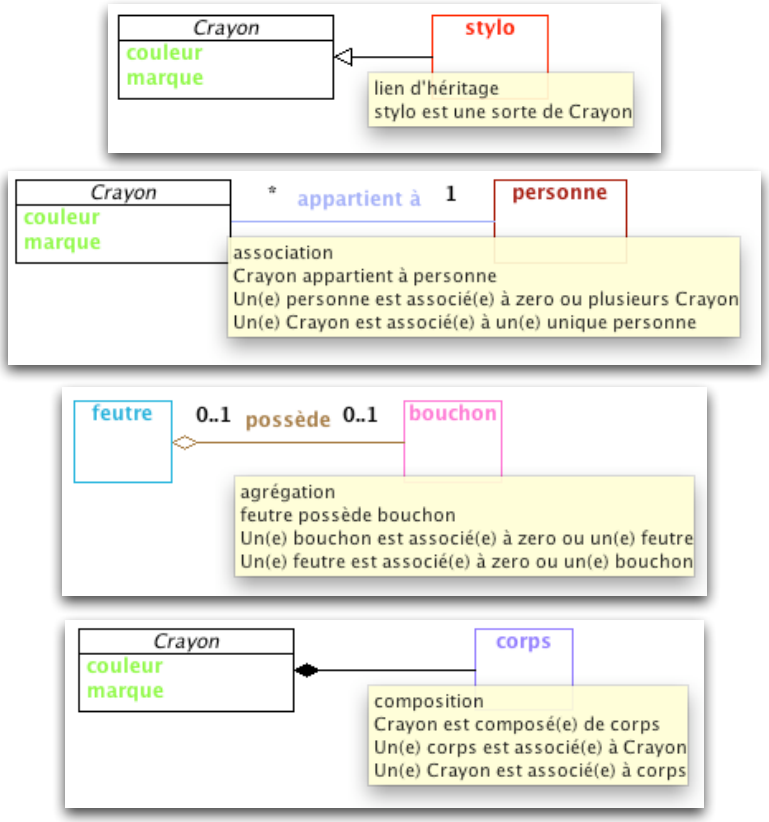


FIG. *5.10 – Exemples de reformulation de relations dans* DIAGRAM

5.1.2.3 Reformulation lors de l'édition des relations

Dans DIAGRAM, il est possible d'éditer les relations (sauf l'héritage et la réalisation) afin de les modifier. Pour cela, il suffit de double-cliquer sur une classe ou une relation. La fenêtre d'édition d'une relation se compose d'un onglet contenant un champ de texte pour l'édition du nom de la relation, et dans les cas des associations, agrégations et compositions, d'un onglet pour les multiplicités ainsi que d'un onglet pour les rôles.

Durant les expérimentations, nous avons constaté que les étudiants éprouvaient des difficultés à indiquer les multiplicités des relations. En particulier, nous avons souvent constaté qu'ils inversaient les multiplicités de chaque « côté » de la relation. Nous avons donc ajouté une aide supplémentaire rendue possible par DIAGRAM et utilisant la reformulation des relations. Nous avons précisé, dans les fenêtres d'édition, les classes concernées par la multiplicité de chaque « côté » de la relation. Pour les onglets correspondant au nom et aux multiplicités, nous affichons également un champ de texte

correspondant à la reformulation du nom et des multiplicités des relations. Dès que le nom ou qu'une multiplicité est modifié, le message de reformulation est automatiquement mis à jour. Cela permet à l'apprenant de visualiser la reformulation de la relation et des multiplicités immédiatement. La figure 5.11 montre un exemple de reformulation du nom et des multiplicités d'une relation.

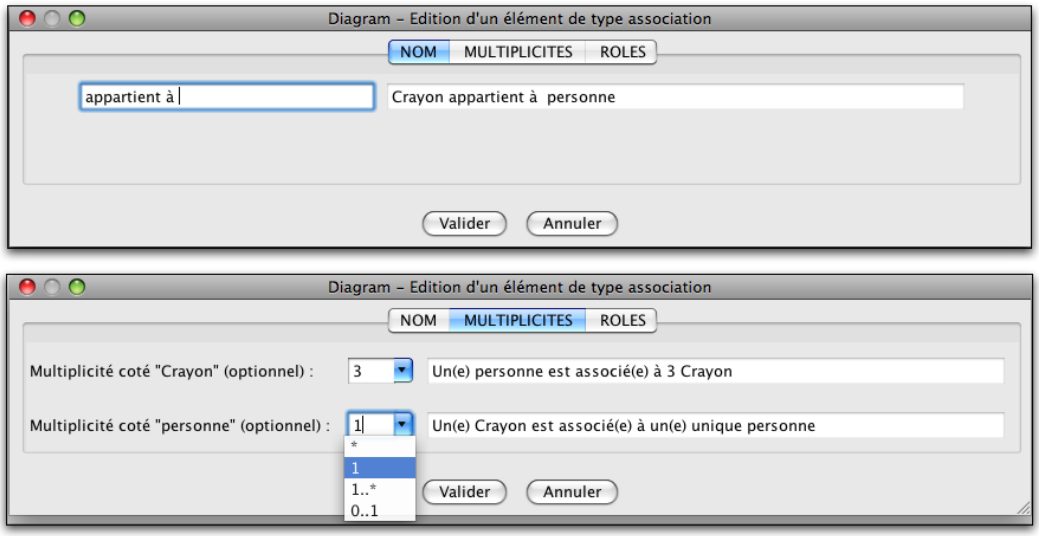


FIG. *5.11 – Exemples de reformulation à l'édition d'une association dans* DIAGRAM

L'édition des multiplicités est réalisée grâce à une liste déroulante proposant les quatre valeurs par défaut : *, 1, 1..* et 0..1 pour lesquelles nous utilisons la reformulation usuelle, présentée dans le chapitre 2, tableau 2.1, page 21. Toutefois, il est possible d'entrer une autre valeur qui sera alors utilisée dans les messages de reformulation comme le montre la figure 5.11.

5.1.3 Rétroactions

Ce paragraphe décrit l'implémentation des rétroactions dans DIAGRAM. C'est pourquoi nous décrivons dans un premier temps la méthode de diagnostic utilisée dans DIAGRAM. Cette méthode liste les différences structurelles entre un diagramme de l'apprenant et un diagramme de référence. Nous avons converti ces différences structurelles en différences pédagogiques selon notre taxonomie présentée dans le chapitre 4, paragraphe 4.2.4.2, page 77. Nous présentons donc ensuite la correspondance que nous établissons entre ces deux taxonomies de différences. Enfin nous détaillons l'affichage des rétroactions dans DIAGRAM.

5.1.3.1 Méthode de diagnostic dans DIAGRAM

Ce paragraphe présente la méthode de diagnostic de diagrammes de classes élaborée par Ludovic Auxepaules au cours de son travail de doctorat [Auxepaules et al., 2007] et implémentée dans DIAGRAM.

L'approche adoptée consiste à comparer le modèle de l'apprenant à un modèle de référence, en tentant de les apparier [Py et al., 2008]. Afin d'exploiter les particularités des diagrammes UML, la méthode de diagnostic utilise des structures caractéristiques appelées *motifs* qui correspondent aux différents niveaux de granularité d'un modèle

[Auxepaules, 2008]. Les motifs simples représentent les éléments du modèle. Ces motifs sont organisés et structurés en motifs complexes que sont les chaînes d'associations et les hiérachies d'héritages. Ces motifs complexes sont exploités pour guider le diagnostic et apparier les différentes parties du diagramme à un niveau de granularité élevé, avant d'apparier les éléments les plus simples. L'algorithme de diagnostic utilise des fonctions de similarité qui calculent un score pour chaque couple de motifs comparés. Les algorithmes de comparaison et d'appariement des diagrammes sont descendants (*top-down*) : les motifs complexes, puis les motifs simples, sont comparés et les paires sont classées par scores croissants. L'algorithme d'appariement des diagrammes produit en sortie des paires de motifs totalement ou partiellement appariés. Les appariements partiels sont étiquetés avec le type de différence qui les caractérise, selon la taxonomie présentée figure 5.12 des différences structurelles (notée TDS).

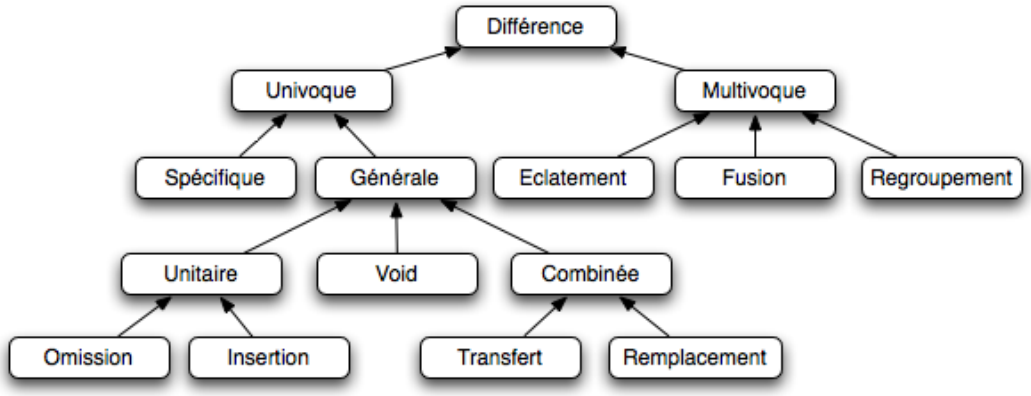


FIG. *5.12 – Taxonomie des différences structurelles (tirée de [Auxepaules et al., 2007])*

Une différence *univoque* est repérée lorsque l'appariement porte sur exactement un motif de chaque diagramme. Parmi ces différences univoques, les différences *spécifiques* sont relatives aux propriétés et à la sémantique des motifs. Pour les motifs simples, elles concernent le nom, la visibilité, l'abstraction, l'agrégation ou l'orientation des relations. Les différences spécifiques des motifs complexes sont relatives à la propagation des propriétés et des relations d'une classe mère vers ses filles et inversement. Les différences *générales* décrivent l'organisation et la structuration des motifs dans les diagrammes. Les deux différences de base sont l'*omission* et l'*insertion* d'un motif. Ces différences peuvent être combinées pour former des différences encore plus complexes, comme le *transfert* (déplacement d'un élément d'un endroit vers un autre) ou le *remplacement* (substitution d'un élément par un autre, au même endroit).

Une différence *multivoque* se produit quand un motif d'un diagramme est apparié à un groupe de motifs de l'autre diagramme. On distingue trois cas : un motif du diagramme d'origine peut être apparié avec plusieurs motifs de l'autre diagramme (*éclatement*), ou inversement (*fusion*), et un groupe de motifs du diagramme d'origine peut s'apparier à un autre groupe de motifs du second diagramme (*regroupement*).

5.1.3.2 Correspondance entre les taxonomies structurelles et pédagogiques

Les différences entre le diagramme de l'apprenant et le diagramme de référence sont établies à un niveau structurel dans DIAGRAM. Afin de générer les rétroactions basées sur notre taxonomie de différences pédagogiques (présentée dans le chapitre 4,

paragraphe 4.2.4.2, page 77), il est donc nécessaire d'établir une correspondance entre les différences structurelles et les différences pédagogiques. Nous avons donc comparé les sorties du diagnostic et les différences simples que nous avions identifiées afin de vérifier leur concordance. Cela nous a permis d'établir un tableau général de correspondance entre les deux taxonomies (cf. tableau 5.1).

TAB. *5.1 – Correspondance entre les deux taxonomies*

Différences structurelles (TDS)	**Différences pédagogiques (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*

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. Nous n'avons trouvé aucune correspondance de cette différence dans la taxonomie des différences pédagogiques. La catégorie des différences univoques spécifiques regroupe les différences relatives aux propriétés et à la sémantique des motifs. Dans la taxonomie des différences pédagogiques, nous retrouvons 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. 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 taxonomie des différences structurelles, correspond à un transfert dans la taxonomie des différences pédagogiques. Les différences non factorisées de remplacement et de type *void* expriment le fait que les motifs sont appariés dans les diagrammes comparés et n'ont donc pas de correspondance dans la taxonomie des différences pédagogiques.

5.1.3.3 Affichage des rétroactions dans DIAGRAM

Comme nous l'avons décrit dans le chapitre 4, paragraphe 4.2.4.2, page 77, les rétroactions sont construites à partir des différences pédagogiques simples et composées. Elles se présentent sous la forme de messages textuels associés à une ou plusieurs modalités parmi l'indication, la question et la proposition. Dans DIAGRAM, nous avons choisi une représentation arborescente pour afficher les rétroactions. En effet, la structure arborescente nous semble adaptée pour l'affichage imbriqué des éléments contenus dans les rétroactions. Chaque rétroaction est un « nœud remarque » numéroté. Ces remarques sont classées en trois catégories : éléments à corriger, éléments ajoutés et éléments omis. Les éléments

ajoutés et les éléments omis regroupent respectivement les cas d'ajout (ajout d'une classe, d'une relation, etc.) et d'omission. La catégorie des éléments à corriger regroupe les rétroactions liées à toutes les autres différences pédagogiques. Cette organisation améliore la lisibilité des messages et indique à l'apprenant la nature générale de chaque remarque. Les trois catégories représentent les nœuds de plus haut niveau dans l'arborescence, et chaque catégogie contient autant de fils que de « nœuds remarque » dans cette catégorie. Nous proposons d'ajouter pour chaque « nœud remarque » une boîte à cocher afin que l'apprenant puisse noter quelles sont les remarques qu'il a pris en compte. Chaque « nœud remarque » contient un fils représentant la modalité de plus haut niveau, définie pour cette remarque. Si plusieurs modalités sont définies, nous avons choisi de les afficher de manière imbriquée plutôt que de les afficher au même niveau afin d'éviter que l'apprenant soit tenté d'accèder immédiatement à la modalité la plus précise. Nous pensons qu'il est préférable que l'apprenant accède par étape à chacune des modalités, chaque étape permettant une évaluation de sa production. Ainsi, le message d'une modalité n'est visible que si le message de la modalité précédente a été affiché. Par exemple, si une remarque est composée d'une indication et d'une question, il est nécessaire d'afficher en premier l'indication avant de pouvoir afficher la question. La figure 5.13 montre un exemple d'affichage de plusieurs rétroactions lorsque l'arbre est entièrement déplié.

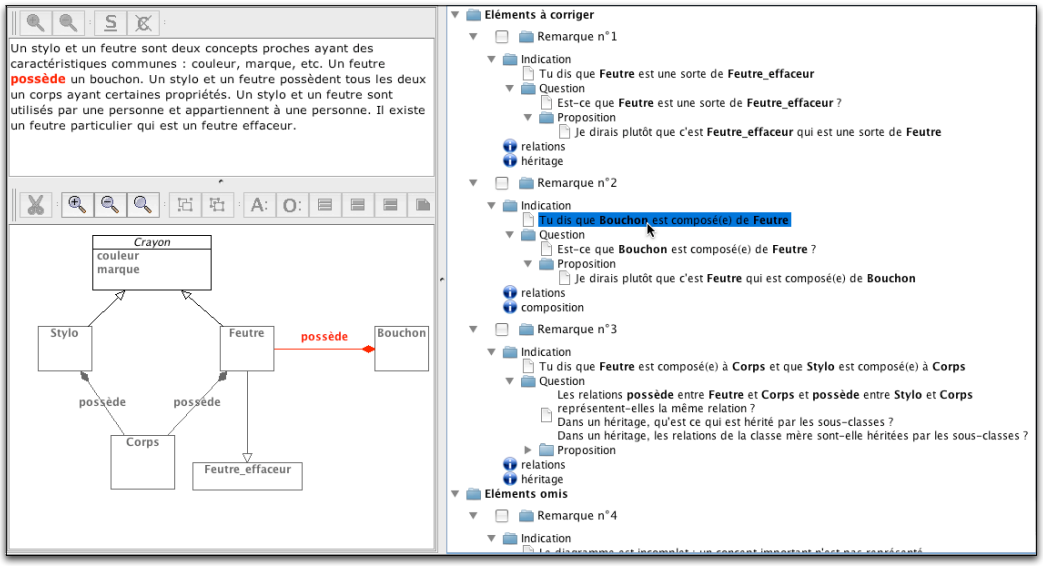


FIG. *5.13 – Exemple de rétroactions dans* DIAGRAM

Au cours des expérimentations des rétroactions intégrées dans DIAGRAM, nous avons testé deux affichages de l'arborescence des rétroactions. Dans le premier cas, les « nœuds remarque » sont initialement fermés et l'apprenant doit cliquer pour faire apparaître les messages. Dans le second cas, le message de la modalité la plus générale de chaque « nœud remarque » est visible dès le premier affichage de l'arborescence des rétroactions. Nous avons testé la seconde solution afin d'inciter encore davantage les apprenants à lire les remarques proposées, mais cette solution rend l'interface plus chargée à première vue.

Afin de faciliter la vérification du diagramme au cours de la lecture des messages de rétroaction, nous mettons en évidence les éléments graphiques concernés (et les éléments textuels auxquels ils sont liés s'ils existent) par le message de rétroaction sélectionné à un moment donné. Pour cela, nous avons choisi d'utiliser la couleur rouge pour colorier les

éléments du graphique et les éléments du texte liés au message sélectionné, et d'afficher tous les autres éléments en noir. Dans l'exemple présenté figure 5.13, le nœud *Indication* de la remarque n°2 est sélectionné. Celle-ci concerne la composition 'possède' entre les classes 'Bouchon' et 'Feutre'. La relation apparaît en rouge dans l'espace graphique ainsi que l'expression 'possède' à laquelle elle est liée dans l'éditeur textuel. Tous les autres éléments textuels ou graphiques sont affichés en noir.

Enfin, nous avons associé à chaque « nœud remarque » une ou plusieurs pages d'un cours de modélisation orientée objet (élaboré par Thierry Lemeunier), susceptibles d'aider l'apprenant dans le processus d'auto-correction de son diagramme. La figure 5.14 montre l'exemple d'une page de cours introduisant la notion d'héritage.

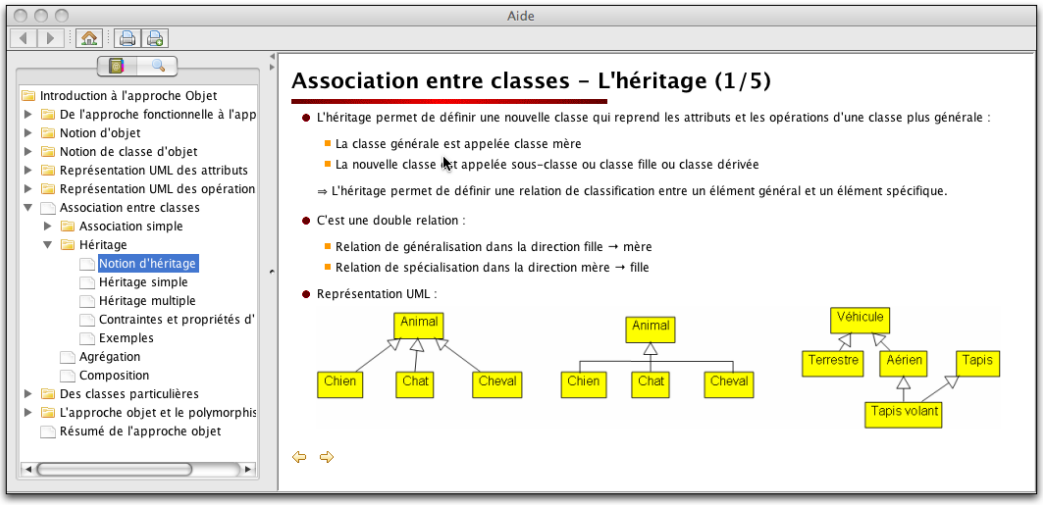


FIG. *5.14 – Rappel de cours dans* DIAGRAM

5.1.4 Activités scénarisées dans DIAGRAM

Ce paragraphe présente le déroulement des activités dans DIAGRAM. Ces activités sont scénarisées selon les cinq phases : lecture, modélisation, relecture, rétroactions et modélisation avec visualisation des précédents messages de rétroaction. Nous présentons tout d'abord les élément communs du déroulement des trois types d'activités que nous avons intégrés dans DIAGRAM, puis nous détaillerons les cas de l'activité de type « Diagramme à créer ». Enfin nous décrirons l'étape de lecture des activités « Diagramme à compléter » et « Diagramme à corriger » car elle diffère de celle de l'activité « Diagramme à créer ».

5.1.4.1 Déroulement global des activités et intégration de contraintes

Toutes les activités se déroulent selon le schéma présenté dans le chapitre 4, figure 4.7, page 76. Lorsque l'apprenant utilise DIAGRAM il peut sauvegarder son travail à tout moment. Nous avons choisi de n'autoriser la fermeture du logiciel qu'après l'étape 3, car nous considérons que l'étape de vérification doit être effectuée au moins une fois. Si l'apprenant quitte l'exercice lors de l'étape 1, il pourra reprendre à cette étape. S'il quitte l'exercice à l'étape 3, 4 ou 5 c'est l'étape 2 qui sera automatiquement proposée à la reprise de l'exercice. Des allers et retours sont possibles entre les étapes 2 et 3, ainsi qu'entre les

étapes 4 et 5. Nous avons également défini deux contraintes dans la scénarisation globale des activités :

- contrainte de soulignage d'expressions : pour un exercice donné, l'enseignant peut spécifier un ensemble d'expressions devant être soulignées. Le passage de la première à la deuxième étape n'est alors autorisé que si toutes les expressions de la liste ont été soulignées. Cette contrainte a été ajoutée car nous avons constaté lors d'expérimentations que certains apprenants passaient trop rapidement cette étape. Elle a pour objectif de s'assurer que l'apprenant a correctement lu l'énoncé, et a effectivement repéré les mots jugés importants par l'enseignant.
- contrainte de lecture ou relecture complète : les étapes utilisant l'outil de passage de la souris ne peuvent être validées que si elles ont été réalisées en entier, c'est-à-dire si l'apprenant a entièrement parcouru le texte de l'énoncé avec la souris. Dans le cas contraire, un message d'erreur est affiché. Cette contrainte a notamment été ajoutée pour s'assurer que les apprenants vérifient leur diagramme en entier dans l'étape de relecture, ce qui n'était pas le cas de certains apprenants lors des expérimentations.

Les activités se différencient par les objectifs des étapes et les éditeurs présents en particulier pour l'étape de lecture. Nous détaillons ci-dessous un exemple de déroulement complet de l'activité de « Diagramme à créer », ainsi que l'étape de lecture pour les scénarios de « Diagramme à compléter » et « Diagramme à corriger ».

5.1.4.2 Diagramme à créer

Etape de lecture

Le scénario de diagramme à créer commence par une étape de lecture. L'interface ne contient que l'énoncé, comme le montre la figure 5.15 et un bouton de validation permettant de passer à l'étape suivante. L'apprenant souligne les expressions importantes pour la modélisation et passe à l'étape suivante lorsqu'il a terminé.

FIG. *5.15 – Etape de lecture du scénario « Diagramme à créer » dans* DIAGRAM

Etape de modélisation

Dans l'étape de modélisation, l'interface contient l'énoncé, l'espace de modélisation et un bouton de passage à l'étape de relecture comme le montre la figure 5.16. L'apprenant modélise son diagramme en se basant sur les mots qu'il a repérés à l'étape de lecture. Il dispose dans cette étape des outils que nous avons présentés auparavant, comme le soulignage, le coloriage et le rattachement, et des aides métacognitives de création et de reformulation des éléments.

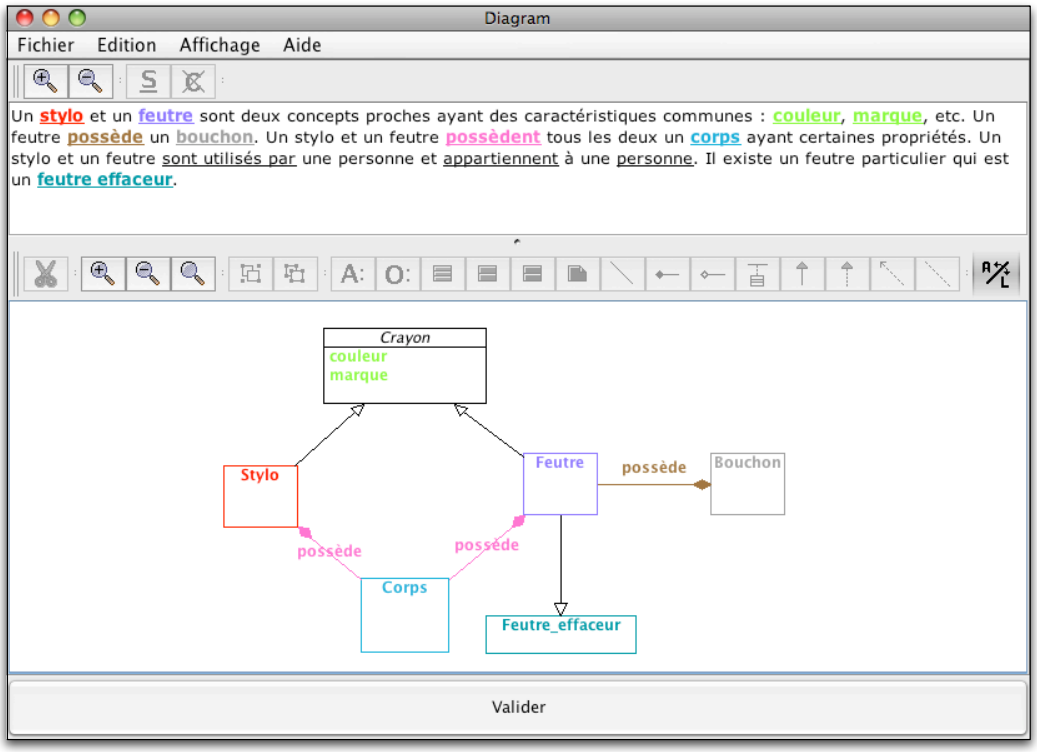


FIG. *5.16 – Etape de modélisation du scénario « Diagramme à créer » dans* DIAGRAM

Lorsque l'apprenant estime avoir terminé son diagramme, il peut passer à l'étape de relecture.

Etape de relecture

La figure 5.17 montre l'interface au commencement de l'étape. L'énoncé apparaît en noir et les couleurs des éléments graphiques sont masquées sauf pour les éléments créés en mode libre et le contour des classes auxquelles ils sont liés, dans le cas des relations. Au fur et à mesure du passage de la souris sur le texte, les éléments textuels coloriés et les éléments graphiques correspondants reprennent leurs couleurs. Dans cette étape, il est possible à tout moment de retourner à l'étape de modélisation. En revanche, il faut avoir réalisé l'étape de relecture en entier, c'est-à-dire avoir parcouru tout le texte à l'aide de la souris, pour accéder à l'étape suivante.

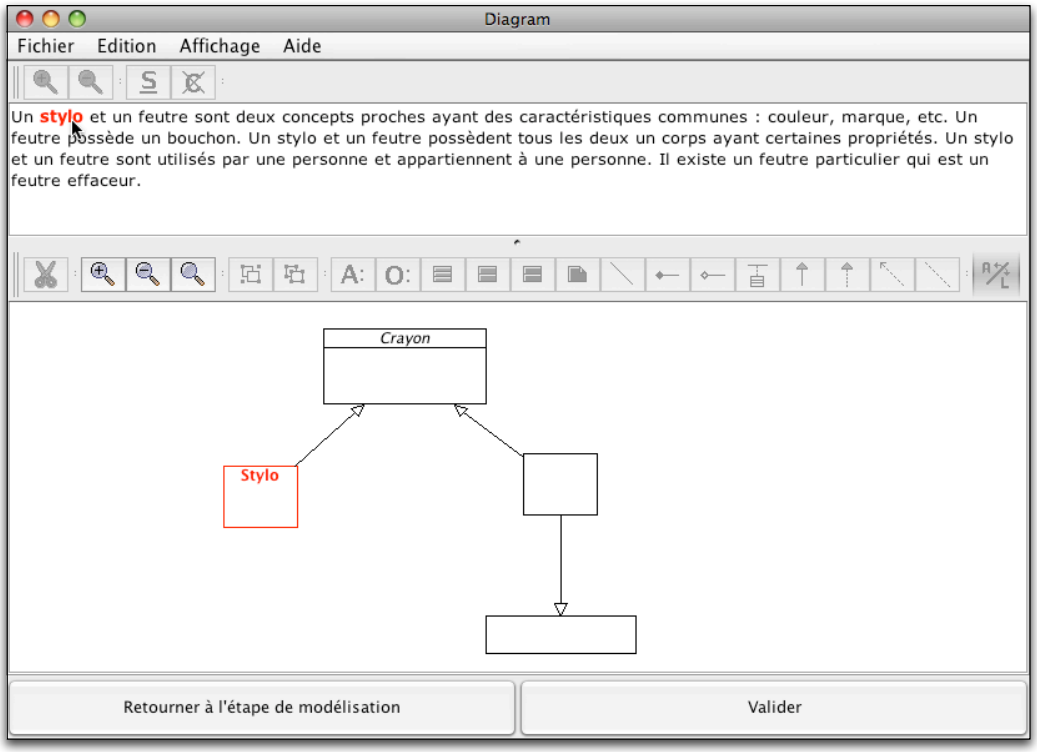


FIG. *5.17 – Etape de relecture du scénario « Diagramme à créer » dans* DIAGRAM

Etape de rétroactions

La figure 5.18 montre l'interface au début de l'étape de rétroactions. Tous les messages sont fermés, et l'apprenant doit cliquer pour les faire apparaître. Lorsqu'un message est sélectionné, les éléments graphiques et textuels correspondants sont mis en évidence grâce à la couleur rouge. Dans cette phase il n'est pas possible d'éditer le diagramme mais les messages de reformulation des éléments graphiques sont toujours affichés afin de fournir une aide supplémentaire à l'apprenant lors du processus de contrôle de son diagramme. Si l'apprenant veut modifier son diagramme, il doit passer à une nouvelle étape de modélisation.

Etape de modélisation avec visualisation des précédentes rétroactions

Après avoir confronté les messages fournis dans l'étape de rétroactions avec sa propre modélisation, l'apprenant a la possibilité de modifier son diagramme. Pour y parvenir, il accède à une cinquième étape. Dans DIAGRAM, l'interface de cette étape est similaire à celle de l'étape de rétroactions mais les éléments textuels et graphiques ont repris leur couleur et les éditeurs sont actifs : le diagramme est modifiable. Les boîtes à cocher associées aux remarques peuvent être utilisées par l'apprenant pour noter les remarques prises en compte. Par exemple, dans la figure 5.19, l'apprenant a pris en compte les remarques 1 et 3 et a modifié son diagramme en conséquence. En revanche, il n'a pas (ou pas encore) tenu compte des remarques 2 et 4. Afin de conserver la liberté de l'apprenant d'évaluer sa propre solution, nous n'imposons pas de prendre en compte les remarques. L'apprenant peut donc quitter l'exercice à tout moment durant les étapes 4 et 5. Les fonctionnalités de sauvegarde et de chargement des exercices permettent à l'apprenant de

retravailler par la suite sur un exercice s'il le désire.

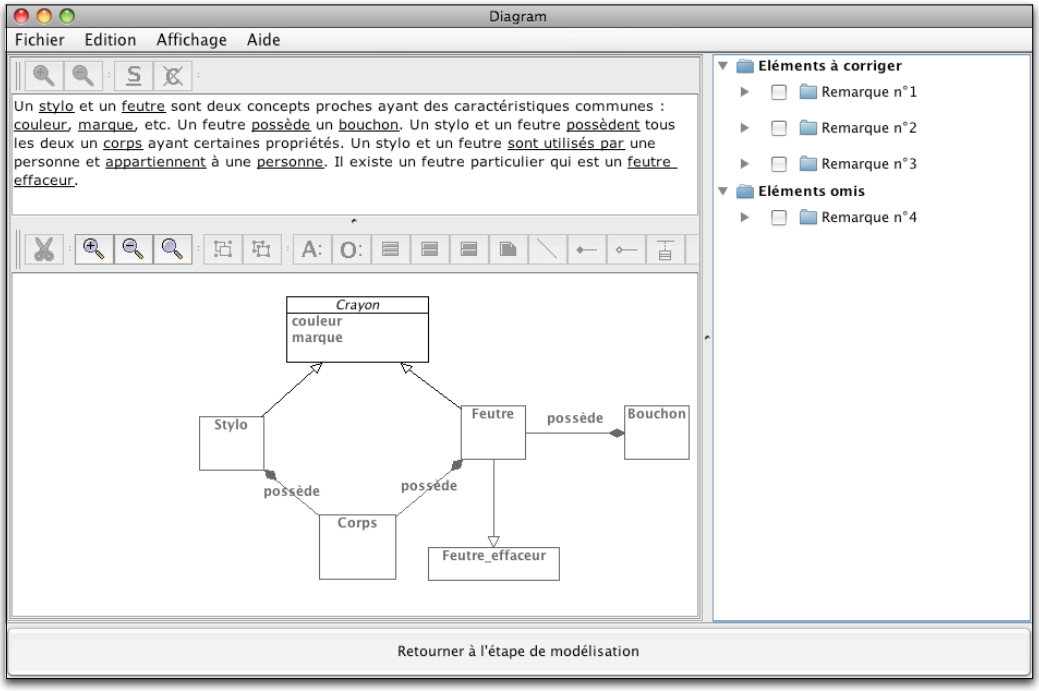


FIG. *5.18 – Etape de rétroactions du scénario « Diagramme à créer » dans* DIAGRAM

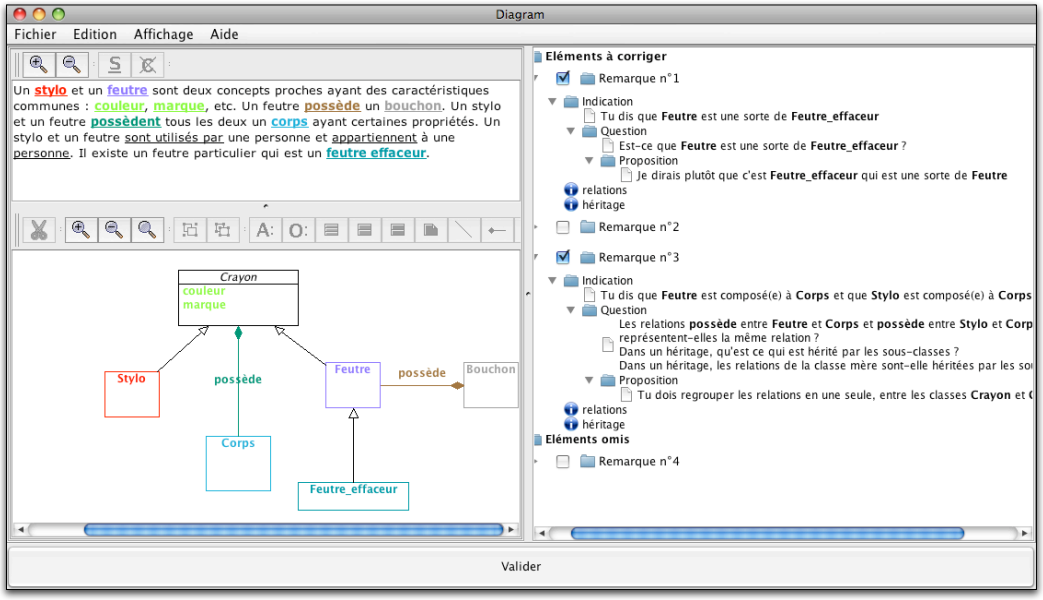


FIG. *5.19 – Etape de modélisation avec visualisation des prédécentes rétroactions du scénario « Diagramme à créer » dans* DIAGRAM

5.1.4.3 Diagramme à compléter

Cette activité consiste à compléter une ébauche de diagramme. Dans la première étape, l'apprenant doit lire l'énoncé et analyser l'ébauche de diagramme donnée.

Etape de lecture

Nous utilisons dans cette étape l'outil de passage de la souris afin de faciliter le repérage des éléments du texte modélisés dans l'ébauche du diagramme. Pour cela, l'interface contient les éditeurs textuels et graphiques. Au départ de l'étape, l'éditeur graphique ne contient que les éléments de l'ébauche qui ne sont pas liés au texte, comme le montre la figure 5.20.

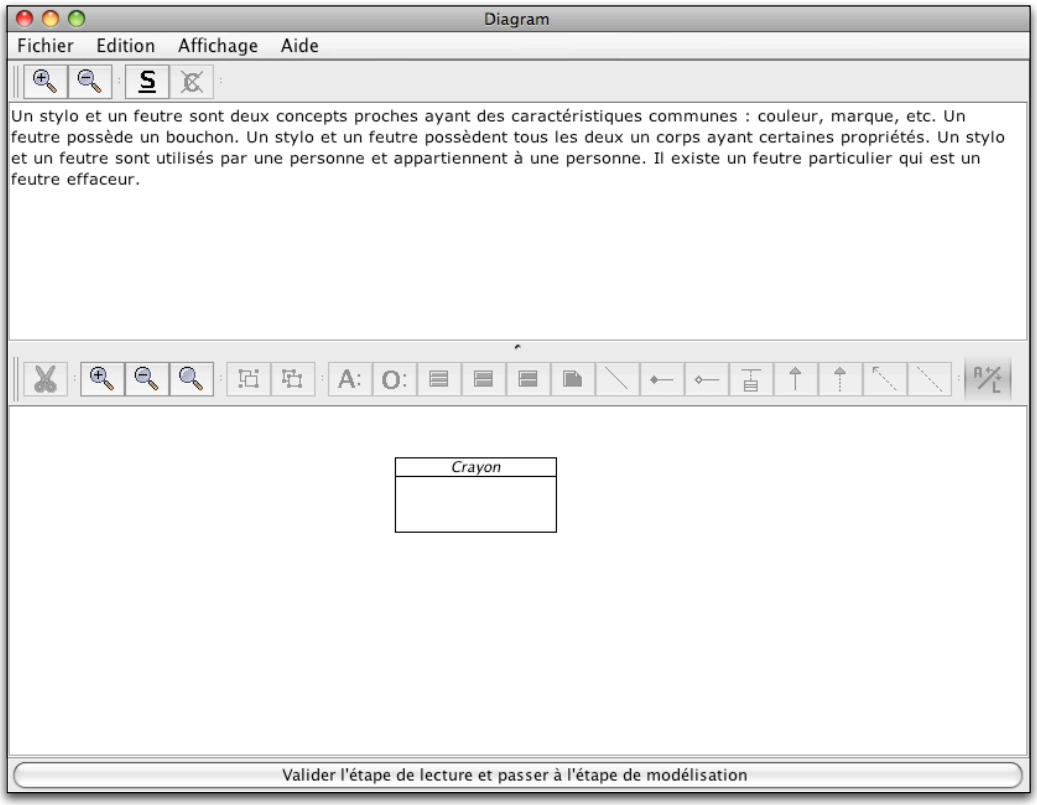


FIG. *5.20 – Début de l'étape de lecture du scénario « Diagramme à compléter » dans* DIAGRAM

L'apprenant passe la souris sur le texte et les éléments de l'ébauche de diagramme apparaissent au fur et à mesure. Dans cette étape, l'apprenant ne peut pas modifier le diagramme, mais accède à l'outil de soulignage pour repérer des mots qu'il utilisera pour compléter le diagramme. La figure 5.20 montre l'interface de DIAGRAM à la fin de l'étape. L'apprenant peut passer à l'étape de modélisation lorsqu'il a parcouru le texte en entier.

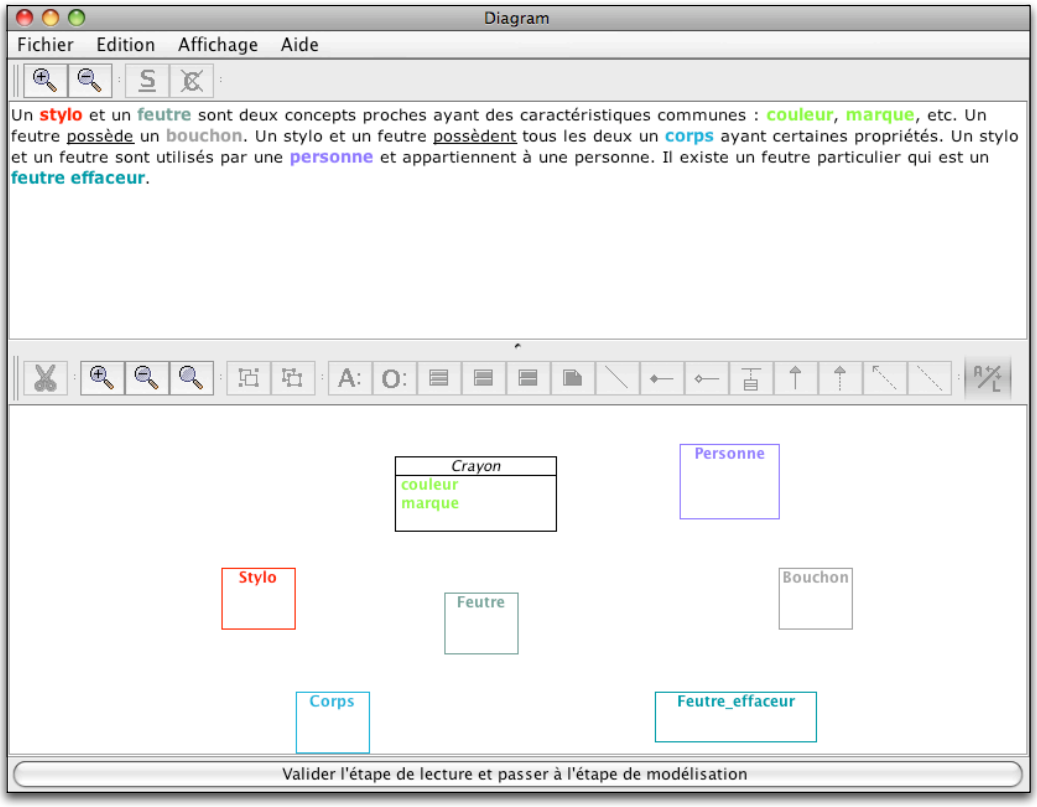


FIG. *5.21 – Fin de l'étape de lecture du scénario « Diagramme à compléter » dans* DIAGRAM

5.1.4.4 Diagramme à corriger

Cette activité consiste à corriger un diagramme dans lequel des erreurs de modélisation ont été ajoutées. Dans la première étape, l'apprenant doit lire l'énoncé et analyser le diagramme fourni afin de repérer les erreurs de modélisation.

Etape de lecture

Comme pour l'activité de diagramme à compléter, nous utilisons l'outil de passage de la souris afin de faciliter le repérage des éléments du texte modélisés dans le diagramme. Au départ de l'étape, l'éditeur graphique ne contient que les éléments du diagramme qui ne sont pas liés au texte, comme le montre la figure 5.22.

L'apprenant passe la souris sur le texte et les éléments du diagramme apparaissent au fur et à mesure. Cela permet à l'apprenant de repérer comment chaque élément est modélisé, et de repérer les erreurs de modélisation. Durant cette étape l'apprenant ne peut pas modifier le diagramme, mais accède à l'outil de soulignage. De plus, les messages de reformulation au passage du curseur sont disponibles dans cette étape. La figure 5.22 montre l'interface de DIAGRAM à la fin de l'étape. L'apprenant peut passer à l'étape de modélisation lorsqu'il a parcouru le texte en entier.

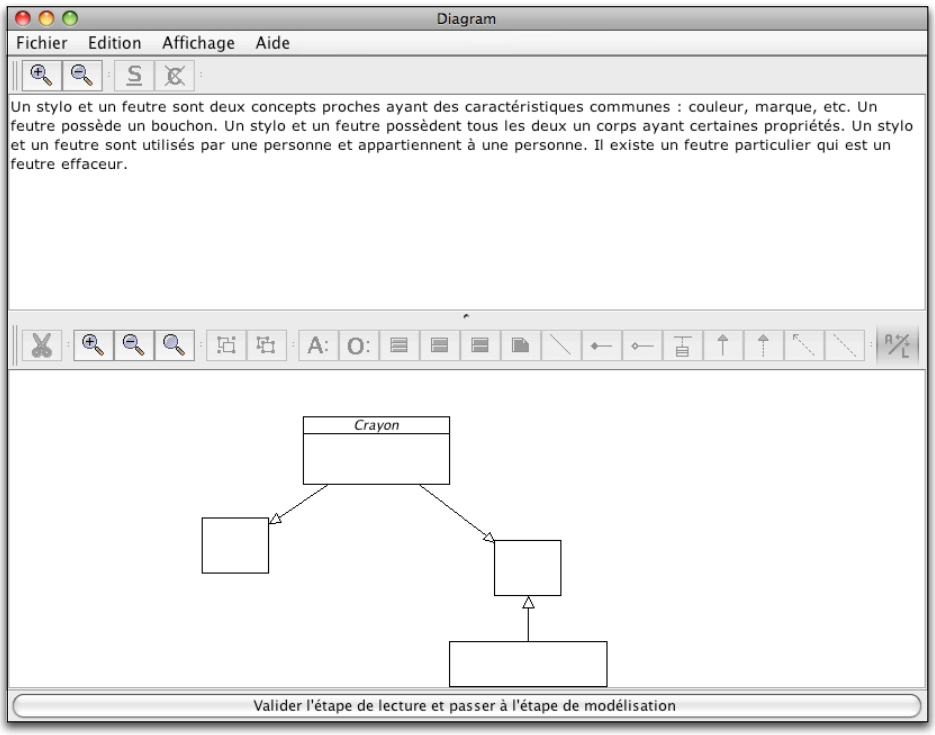


FIG. *5.22 – Début de l'étape de lecture du scénario « Diagramme à corriger » dans* DIAGRAM

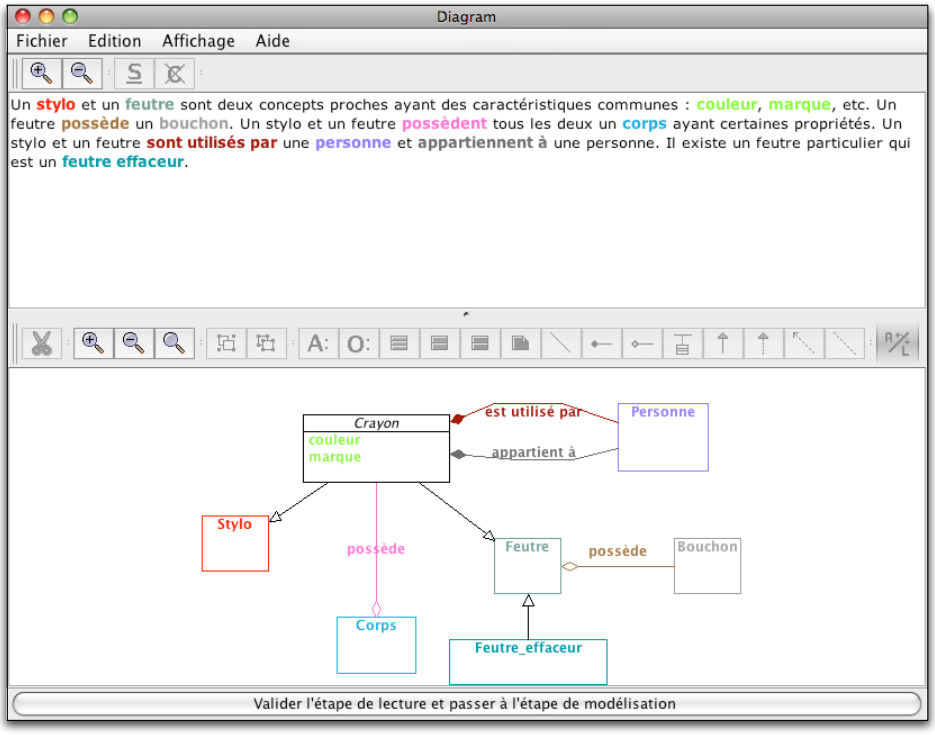


FIG. *5.23 – Fin de l'étape de lecture du scénario « Diagramme à corriger » dans* DIAGRAM

5.2 Réalisation informatique : présentation du logiciel DIAGRAM

DIAGRAM est un EIAH pour l'apprentissage de la modélisation orientée objet qui a été conçu et développé au sein du projet « Interaction et connaissances dans les EIAH pour la modélisation ». Il constitue le support de l'implémentation de notre modèle d'interaction. Nous présentons dans cette section l'architecture générale de DIAGRAM et nous détaillons certains aspects de son implémentation. En particulier, nous précisons nos différentes contributions en terme de réalisation informatique. Dans un dernier paragraphe nous présentons les spécifications d'une version « enseignant » de DIAGRAM permettant aux enseignants de créer des exercices.

5.2.1 Architecture générale de DIAGRAM

DIAGRAM a été développé en Java, sous l'environnement Eclipse. Il est constitué de différents modules selon l'architecture présentée figure 5.24. Certains composants complexes sont eux-mêmes constitués de sous-composants. Du point de vue de la programmation Java, chaque composant est codé dans un package séparé et chaque sous-composant est inclus dans un package englobant.

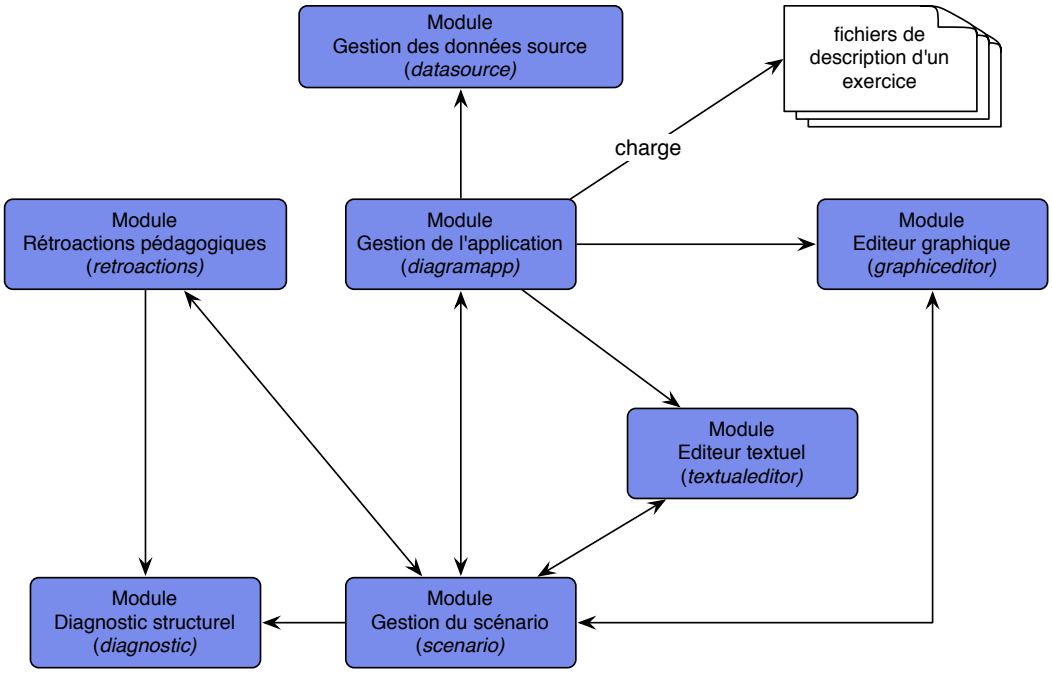


FIG. *5.24 – Architecture générale de* DIAGRAM

DIAGRAM est constitué de 7 composants principaux :

- le module de **gestion de l'application** est chargé de l'affichage et de la gestion de la fenêtre principale de l'application ;
- le module de l'**éditeur textuel** traite l'affichage et de la gestion de l'énoncé de l'exercice ;
- le module de l'**éditeur graphique** gère l'espace de construction du diagramme de classes ;

- le module de **gestion du scénario** est en charge du déroulement global des activités scénarisées ;
- le module de **diagnostic** gère la comparaison du diagramme de l'apprenant avec le diagramme de référence ;
- le module des **rétroactions** est en charge de la production et de l'affichage des rétroactions ;
- le composant de **gestion des données source** fournit des services tels que la lecture et l'interprétation des éléments de modélisation disponibles dans l'application et l'exportation du diagramme au format image.

Une version préliminaire de DIAGRAM avant le déroulement de nos travaux de recherche, avait été développée. Elle contenait tous les modules à l'exception du module de scénario et du module de rétroactions que nous avons entièrement implémentés, ainsi que du module de diagnostic réalisé par Ludovic Auxepaules. Cette version contenait les étapes figées du scénario de diagramme à créer, la structuration interne des éléments des éditeurs textuels et graphiques, et une première implémentation des outils de soulignage, coloriage et rattachement dont nous avons modifié et affiné le fonctionnement. Nous sommes intervenus dans tous les modules existants sauf le module de gestion des données source[1]. Nous détaillons donc le module de gestion de l'application, celui de l'éditeur textuel, celui de l'éditeur graphique, le composant de gestion des scénarios et le module des rétroactions. Nous décrirons également le principe de fonctionnement du module de diagnostic.

5.2.2 Module de gestion de l'application

Ce module est central dans le fonctionnement de DIAGRAM. Il est chargé de fournir la fenêtre principale de l'application qui contient le menu principal et dans laquelle seront inclus les autres composants tels que l'éditeur graphique ou l'éditeur textuel. Lors du lancement de l'application, ce module commande l'ouverture d'une fenêtre de dialogue permettant la saisie de l'identifiant de l'utilisateur. Le menu de la fenêtre principale de DIAGRAM est constitué des items suivant :

- Le menu **Fichier** propose le chargement et l'enregistrement des exercices, la fermeture de l'application, l'exportation en image du diagramme, en interaction avec le package de gestion des données source et le module de scénarios.
- Le menu **Edition** contient les options disponibles d'édition. Les seules implémentées dans le prototype actuel sont les fonctions de suppression, groupement et dégroupement des éléments graphiques.
- Le menu **Affichage** contient les options d'affichage de l'interface comme la modification de la taille de la police d'écriture de l'énoncé ou le zoom sur le diagramme. Enfin, le menu d'aide permet d'accèder aux rappels de cours sur la modélisation orientée objet, dont certaines pages sont proposés dans les rétroactions (cf. paragraphe 5.1.3.3, page 95). Nous avons implémenté dans ce module les liens avec les rappels de cours grâce à l'API JavaHelp [Javahelp]. JavaHelp permet d'intégrer facilement un système d'aide dans une application développée en Java, sous forme d'un ensemble de pages HTML. Cet outil intègre des fonctionnalités telles que la navigation à l'aide d'une table des matières, des index et un outil de recherche.

[1] Une évaluation métrique du développement réalisé dans DIAGRAM est décrite dans l'annexe B, page 143.

Grâce au menu Fichier de cette fenêtre, l'apprenant charge un exercice. Nous avons défini un format de représentation des exercices dans DIAGRAM, que nous présentons ci-dessous.

Représentation d'un exercice

Un exercice est constitué par un fichier compressé, qui contient plusieurs fichiers.
- Le fichier *text.txt* est un fichier texte qui contient l'énoncé de l'exercice.
- Le fichier *scenario.xml* décrit le scénario de l'activité. En l'état actuel, la description externe du scénario est minimale. Elle contient deux éléments : l'étape courante et le type d'activité.
- Le fichier *constraints.txt* contient les contraintes de soulignage, comme nous l'avons décrit dans le paragraphe 5.1.4.1, page 97. Ce fichier est optionnel car la définition de contraintes n'est pas obligatoire.
- Le fichier *avancement.xml* décrit l'état des éditeurs graphiques (éléments graphiques modélisés), expressions soulignées, coloriées, etc. Dans le cas d'une activité de diagramme à compléter ou à corriger, ce fichier décrit l'état initial des éditeurs et sera modifié en fonction des actions de l'apprenant. Dans le cas d'un diagramme à créer, ce fichier n'est pas nécessaire au lancement de l'exercice et sera créé automatiquement lors de la sauvegarde de son travail par l'apprenant.
- Le fichier *reference.xml* décrit le diagramme de référence selon le formlisme UML2. Il sera utilisé par le module de diagnostic.

Suite au chargement d'un exercice, le module de gestion de l'application met en place le scénario d'interaction grâce au module de gestion du scénario.

5.2.3 Module de gestion du scénario

Nous avons entièrement réalisé ce module, qui gère le déroulement d'un exercice en fonction du type d'activité et de l'étape courante. Il s'occupe en particulier du contrôle de la navigation dans l'activité structurée (changement d'étape, arrêt / chargement d'un exercice) et du paramètrage de l'interface (panneaux affichés...). Il réalise également l'interface avec le composant diagnostic en générant les demandes d'évaluation du diagramme de l'apprenant et avec le module de rétroactions en demandant la construction et l'affichage des messages.

Nous avons implémenté le module de gestion du scénario dans DIAGRAM de façon à pouvoir gérer différents types d'exercices, nécessitant chacun un mode d'interaction spécifique. L'ensemble des propriétés et méthodes communes sont regroupés dans une classe abstraite *AbstractScenario* qui définit donc un cadre générique aux scénarios réalisables dans DIAGRAM (figure 5.25). Ces éléments devront être implémentés dans les sous-classes qui représentent les scénarios concrets. Nous définissons également un ensemble de méthodes accessibles aux autres modules du système qui permettront d'interroger le scénario. Par exemple, en cas de demande de fermeture de l'application, le module de gestion de la fenêtre principale demande au scénario s'il est possible de quitter l'application à cet instant. La méthode de gestion de la fermeture de l'application (*iswindowClosingOk*) est implémentée par les scénarios concrets ce qui permet de paramétrer de façon différente chaque scénario indépendamment du reste du système.

Nous avons implémenté trois scénarios concrets, correpondant aux activités que nous avons défini dans le chapitre 4, paragraphe 4.2.1.3, page 72, avec

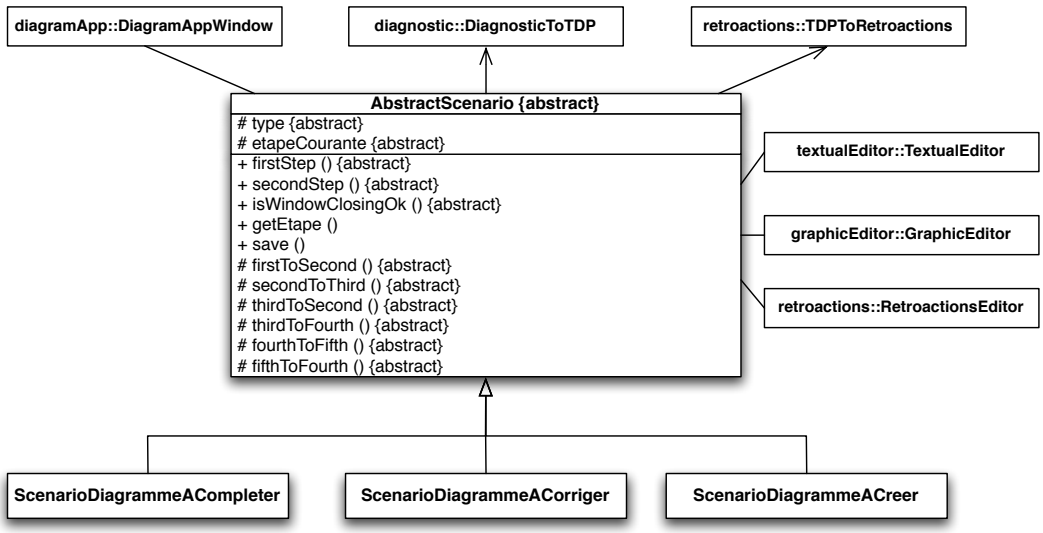


FIG. *5.25 – Architecture du module de gestion du scénario*

les classes *ScenarioDiagrammeACreer*, *ScenarioDiagrammeACompleter* et *ScenarioDiagrammeACorriger*.

5.2.4 Module de l'éditeur de texte

Le module de l'éditeur de texte fournit l'éditeur textuel de l'application. La figure 5.26 montre un extrait de l'architecture de ce composant. Il contient trois classes : *TextGraphicLink*, *TextualEditor* et *TextualEditorToolBar* ainsi que les sous-composants *action* et *texteditor*.

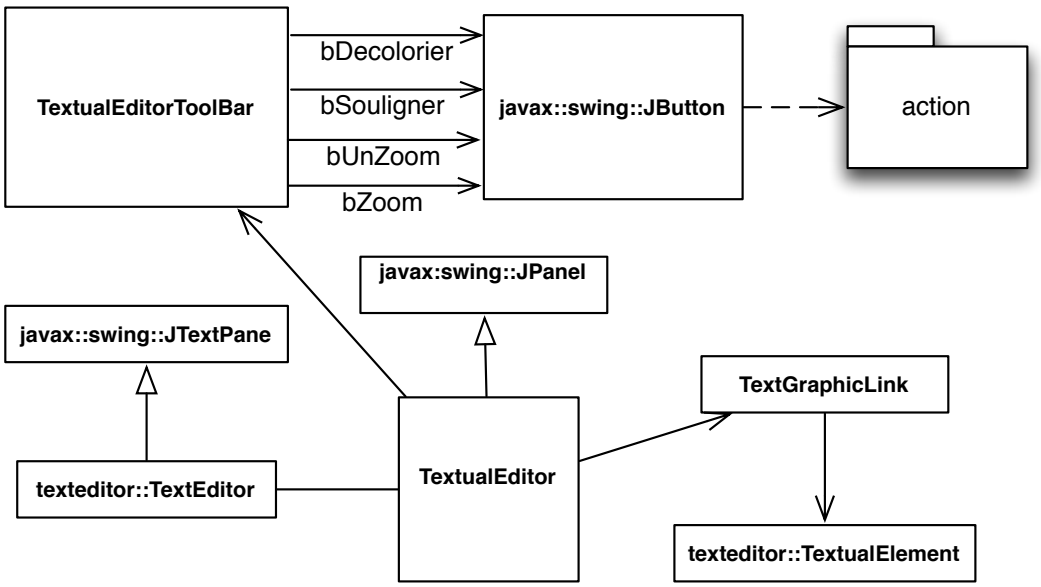


FIG. *5.26 – Architecture du module de l'éditeur textuel*

La classe *TextualEditor* est responsable de la création de l'éditeur de texte lui-même (classe *texteditor.TextEditor*) et de la création de la barre d'outils (classe *TextualEditorToolBar*). Elle gère et maintient à jour l'état de l'éditeur grâce à des listes

correspondant aux expressions soulignées et aux liens créés entre les éditeurs textuel et graphique.

La barre d'outils de l'éditeur est représentée par la classe *TextualEditorToolBar* et l'exécution des actions associées aux boutons de la barre d'outils (en particulier le soulignage) est assurée par le sous-package *action*.

La classe *TextGraphicLink* permet de représenter un lien entre une expression de l'éditeur textuel et un élément graphique ou un lien entre deux expressions textuelles. Ce lien est créé lors de l'utilisation des outils de coloriage ou de rattachement.

Enfin, le package *textualeditor.texteditor* contient la classe *TextEditor* qui fournit le composant de l'éditeur dans lequel l'énoncé de l'exercice est affiché (extension de la classe *javax.swing.JTextPane*). Cet éditeur est capable de lire un fichier au format HTML et de prendre en compte une feuille de style CSS : les couleurs des expressions coloriées de l'énoncé sont alors affichées à l'aide de liens hypertextes.

Ce package contient également les gestionnaires permettant le fonctionnement des différents outils. Le gestionnaire des liens hypertextes identifie le survol par le curseur des liens hypertextes des expressions coloriées et gère la réactivation des couleurs lors de l'utilisation de l'outil de passage de la souris. Le gestionnaire du curseur est en charge du déplacement et de la sélection dans l'éditeur. Nous avons implémenté un gestionnaire des évènements de la souris dans l'éditeur textuel pour la mise en place des facilités de selection en un clic, et l'affichage du menu de rattachement. Enfin le package *textualeditor.texteditor* contient la représentation d'une expression textuelle *TextualElement* du texte présent dans le composant.

5.2.5 Module de l'éditeur graphique

Le module de l'éditeur graphique permet de gérer l'espace graphique de construction du diagramme de classes. Il est lui-même constitué des sous-composants *action* (les actions réalisables dans l'éditeur), *lexicaleditionmodel* (le modèle d'édition), *grapheditor* (l'éditeur de graphe) ainsi que des classes *GraphicEditorToolBar* et *GraphicEditor*, comme le montre la figure 5.27.

La classe *GraphicEditor* est responsable de la création de l'éditeur de graphes (classe *graphiceditor.GraphEditor*), de la création de la barre d'outils (classe *GraphicEditorToolBar*) et du chargement du modèle d'édition lexicale en utilisant le package *datasource*.

Par souci de généricité, les éléments de modélisation qu'il est possible de créer dans DIAGRAM sont décrits dans un « modèle d'édition ». Le modèle d'édition décrit à la fois la manière dont les éléments peuvent être affichés et être édités indépendamment, et la manière dont ces éléments peuvent être liés les uns aux autres afin de former des structures plus complexes. La figure 5.28 présente un extrait de ce modèle.

Pour chaque élément graphique décrit dans ce modèle, un bouton correspondant est créé dans la barre d'outils graphiques. En actionnant ce bouton on délenche la création d'une instance de l'élément graphique correspondant. Les types d'éléments définis sont les éléments unaires, les éléments binaires et les éléments communs. Un élément graphique unaire est un élément pouvant exister indépendamment des autres éléments graphiques. Un élément binaire est un élément reliant deux autres éléments graphiques. Un tel élément ne peut donc pas exister sans les deux autres éléments source et cible. Les éléments communs sont contenus par les éléments unaires.

L'implémentation du modèle est réalisée dans le package *lexicaleditionmodel*, par les

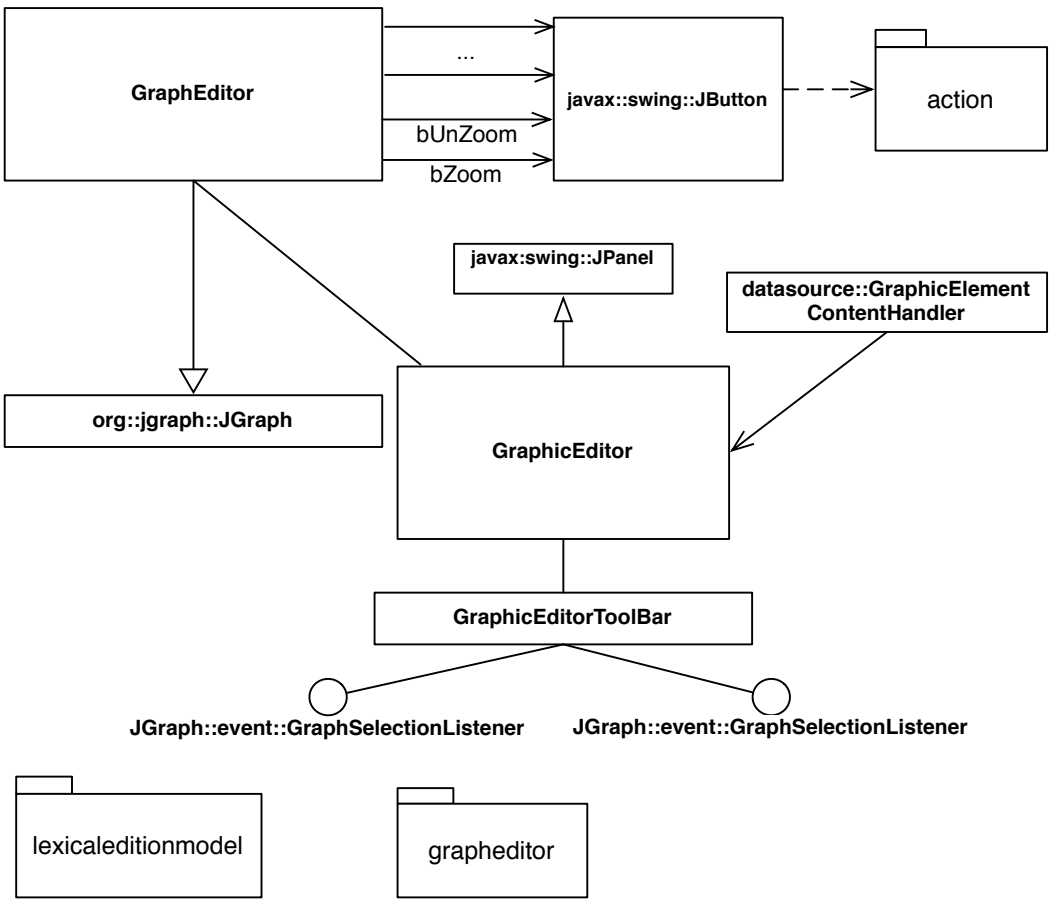


FIG. *5.27 – Architecture du module de l'éditeur graphique*

classes *UnaryGraphicElement*, *BinaryGraphicElement* et *CommonPartGraphicElement*. Les comportements et les attributs communs aux éléments unaires et binaires sont définis dans la classe mère abstraite *AbstractGraphicElement*.

Le package *grapheditor* représente l'éditeur de graphe. Il s'agit d'une extension de l'API JGraph redéfinie en partie pour prendre en compte les spécificités des éléments graphiques souhaités. L'API JGraph fournit essentiellement un composant *JGraph* qui étend la classe *JComponent*. Le composant *JGraph* fonctionne selon le modèle MVC permettant de séparer le modèle (M), la vue du modèle (V) et le contrôle (C) établissant le lien entre le modèle et la vue. Le modèle (M) est un arbre de cellules de deux types : soit un nœud (ou *vertex*), soit un lien (ou *edge*). La classe *GraphEditorMarqueeHandler* est le gestionnaire de la souris de l'éditeur de graphes qui permet notamment de gérer les clics dans l'éditeur lors de l'utilisation des outils de coloriage et de rattachement. Le package *grapheditor* se compose lui-même de quatre packages :

– le package *celltype* définit les cellules utilisées dans l'éditeur de graphes. Elles sont de deux types : les cellules unaires *UnaryGraphicElementCell* qui encapsulent les éléments graphiques unaires, et les cellules binaires *BinaryGraphicElementCell* qui encapsulent les éléments graphiques binaires. Ces classes contiennent l'implémentation des messages de refomulation et d'aide à la création des éléments de l'éditeur graphique grâce à l'API ToolTip. L'avantage de cette API est que l'affichage des tooltips est automatiquement géré par le *ToolTipManager* en fonction de la position du curseur.

```
<?xml version = "1.0" encoding = "UTF-8"?>
[...]
<NORME version = "1.4">
<DIAG_CLASSES>
<COMMON cle = "ATTRIBUT">&attribut;</COMMON>
<COMMON cle = "OPERATION">&operation;</COMMON>
<UNAIRE>
<TYPEUNAIRE [...] img = "Classe.gif" type = "classe">
<NOM [...] valeur = "Classe" />
<ATTRIBUTS [...]>&attribut;</ATTRIBUTS>
<OPERATIONS [...]>&operation;</OPERATIONS>
</TYPEUNAIRE>

<TYPEUNAIRE [...] img = "Classe_Abstraite.gif" motcle =
"Abstract" type = "classe abstraite">
<NOM [...] valeur = "Classe_abstraite" />
[...]
</TYPEUNAIRE>

<TYPEUNAIRE [...] img = "Interface.gif" motcle = "Interface" type
= "classe interface">
<NOM [...] valeur = "Interface" />
[...]
</TYPEUNAIRE>
</UNAIRE>

<BINAIRE>

<TYPEBINAIRE [...] img = "Association.gif" type = "association"
source = "" cible = "">
<NOM [...] />
<MULTIPLICITES[...]>
<MULTIPLICITEDEBUT type = "multiplicité" [...] typevaleur =
"choixmultiple" valeurspardefaut = "*;1;1..*;0..1" [...] />
<MULTIPLICITEFIN type = "multiplicité" [...] typevaleur =
"choixmultiple" valeurspardefaut = "*;1;1..*;0..1" [...] />
</MULTIPLICITES>
</TYPEBINAIRE>

<TYPEBINAIRE [...] img = "Composition.gif" type = "composition"
source = "" cible ="">
<NOM [...] />
</TYPEBINAIRE>
<TYPEBINAIRE [...] img = "Heritage.gif" [...]></TYPEBINAIRE>

</BINAIRE>
</DIAG_CLASSES>
</NORME>
```

FIG. *5.28 – Extrait de la description XML du modèle d'édition*

- le package *cellview* définit les vues des cellules du package *celltype*.
- le package *celleditor* définit les différents éditeurs de cellule. L'édition d'une cellule va permettre l'affichage et la modification de ses valeurs, *via* une fenêtre dédiée. Dans ce package, nous avons implémenté l'affichage des reformulations des relations.
- le package *cellrenderer* définit les afficheurs de vues. Ils sont chargés de l'affichage effectif des vues des cellules.

5.2.6 Module de diagnostic structurel

Le module de diagnostic a été implémenté par Ludovic Auxepaules [Auxepaules et al., 2008]. Pour réaliser la comparaison structurelle entre le diagramme construit par l'apprenant et le diagramme de référence, ce module utilise le diagramme de référence modélisé dans le formalisme UML2. Sur demande du module de scénario d'une évaluation du diagramme de l'apprenant (passage de l'étape 3 à l'étape 4 ou de l'étape 5 à l'étape 4), le module de diagnostic récupère le diagramme courant construit dans l'éditeur graphique et le transforme dans le formalisme UML2. Il peut alors effectuer un diagnostic structurel qui résulte en une liste de différences structurelles entre les deux diagrammes. Les différences structurelles sont ensuite transformées en différences intermédiaires, dotées d'un préfixe, d'un radical et d'un suffixe issus de notre taxonomie des rétroactions de différences pédagogiques présentée dans le chapitre 4, paragraphe 4.2.4.2 page 77. Elles sont ensuite classées en deux listes : les différences simples et les différences composées, chaque différence composée contenant la liste des erreurs associées. Ces différences « intermédiaires » sont préfixées selon la taxonomie des différences pédagogiques mais font directement le lien avec les différences structurelles. Elles permettent d'accèder à l'ensemble des éléments du diagramme de l'apprenant et à l'ensemble des éléments du diagramme de référence que le diagnostic associe à chaque différence. Dans le module de rétroactions, nous utilisons les différences « intermédiaires » pour construire les rétroactions.

5.2.7 Module de rétroactions pédagogiques

Nous avons implémenté entièrement ce module qui construit et organise les rétroactions. La figure 5.29 présente un extrait de l'architecture de ce module.

La classe *TDPToRetroaction* parcourt les résultats de la classe *DiagnosticToTDP* pour créer les rétroactions associées aux différences pédagogiques qu'elle regroupe en trois listes : éléments à corriger, éléments ajoutés et éléments omis.

Nous avons implémenté une classe par rétroaction distincte (parfois la rétroaction est identique, que la différence soit simple ou combinée) dans le sous package *message*. Nous obtenons donc 36 classes réparties sur 9 packages (ajout, dédoublement, dédoublement et transfert, fusion, fusion et transfert, mauvaise représentation, multiplicité, omission, et transfert). Les classes représentant les messages de rétroaction spécialisent de la classe *AbstractRetroaction*. Chaque rétroaction est liée à une instance de la classe *Modalite*, qui contient 3 attributs (indication, question et proposition) et à des pages de cours (optionnel).

La représentation arborescente des rétroactions est implémentée par la classe *ArbreRetroactions*, qui étend la classe *javax.swing.JTree*. Celle-ci construit l'arbre en utilisant différents types de nœuds.

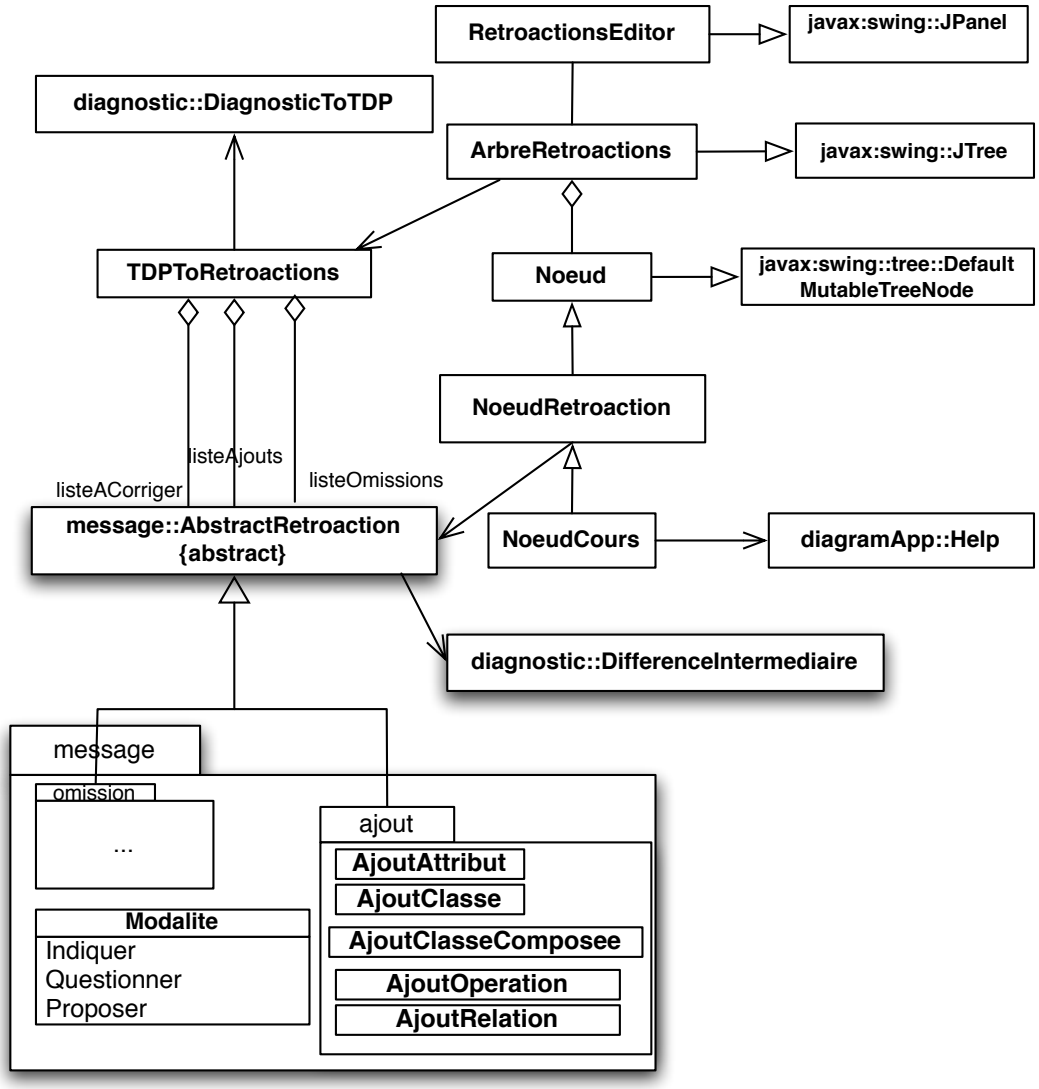


FIG. *5.29 – Architecture du module des rétroactions*

5.2.8 Spécifications d'une version « enseignant » de DIAGRAM

Dans le chapitre 4, paragraphe 4.1.5, page 69, nous avons décrit le rôle de l'enseignant dans la manipulation des concepts de notre modèle. Pour que l'enseignant définisse les exercices à réaliser dans DIAGRAM, il doit donc disposer d'un environnement auteur pour leur création. Dans cet environnement, l'enseignant doit pouvoir :

- écrire l'énoncé de l'exercice. Pour cela, l'environnement doit être flexible et permettre la suppression et l'insertion de texte.
- choisir des contraintes de soulignage, c'est-à-dire les expressions que l'apprenant devra souligner dans l'étape de lecture pour passer à la suite ;
- choisir le type d'activité qu'il construit ;
- créér le diagramme de départ dans le cas des activités de diagramme à compléter et à corriger ;
- créer le diagramme de référence. Ce dernier point est une contrainte due à l'utilisation d'un tel diagramme par le module de diagnostic.

Nous avons développé un prototype d'une version « enseignant » de DIAGRAM. Elle comporte un éditeur de texte entièrement fonctionnel pour la création de l'énoncé et le choix des contraintes de soulignage.

5.3 Conclusion

Ce chapitre présente la mise en œuvre de notre modèle d'interaction dans DIAGRAM. Ces travaux nous ont permis d'en démontrer la faisabilité dans un environnement informatique d'apprentissage. La limite principale à la généricité de nos propositions concerne l'élaboration des rétroactions. En effet, les résultats du système de diagnostic structurel, élaboré dans le cadre du projet « Interaction et connaissances dans les EIAH pour la modélisation » et réifié dans DIAGRAM, nous permettent de déterminer les éléments du diagramme sur lesquels une rétroaction sera apportée. Toutefois, on pourrait envisager de réutiliser les modalités générales d'indication et de question pour inciter l'apprenant à vérifier certains éléments de son diagramme, sans pour autant se baser sur une évaluation comparative du diagramme.

Au cours de nos travaux de recherche, nous avons mené plusieurs expérimentations, permises notamment par l'implémentation de notre modèle dans DIAGRAM. Il s'agissait de vérifier l'utilisabilité du système et d'en valider certains aspects. Ces expérimentations sont présentées dans le chapitre suivant.

CHAPITRE 6

Expérimentations et validation

Sommaire

Résumé

Dans ce chapitre, nous décrivons les différentes expérimentations que nous avons menées. Elles sont de deux types : d'une part des études exploratoires, et d'autre part, des expérimentations permettant de valider nos propositions. Les études exploratoires nous ont permis d'affiner notre modèle. Elles ont consisté en une mise à l'essai d'une version préliminaire de DIAGRAM, et une étude en environnement informatique de nos propositions de tâches d'apprentissage de la modélisation orientée objet. Les expérimentations ont porté sur l'évaluation de DIAGRAM en contexte écologique. L'une consistait en une évaluation globale du système et une analyse de l'effet des rétroactions, alors que la seconde se focalisait sur les aides métacognitives.

6.1 Etudes exploratoires

Nous avons réalisé deux études exploratoires qui nous ont servi à mettre au point et affiner notre modèle. La première est une étude réalisée avec une version préliminaire de DIAGRAM. La seconde est une mise à l'essai en environnement informatique des tâches de « Diagramme à corriger » et « Diagramme à compléter ».

6.1.1 Etudes préliminaires

A l'automne 2005, nous avons réalisé des mises à l'essai de la version préliminaire de DIAGRAM. Elle concernaient une dizaine d'étudiants de seconde année de DEUST ISR de l'université du Maine. Cette étude avait pour objectif d'observer l'utilisation des outils de soulignage et de coloriage en cours de développement et de façon générale, d'observer l'interaction entre les apprenants et le logiciel afin de dégager de nouvelles idées.

Ces mises à l'essai nous ont permis d'affiner le fonctionnement des outils de soulignage, coloriage et rattachement, en particulier les contraintes de fonctionnement pour conserver la cohérence à l'écran. Concernant l'outil de passage de la souris, ces expérimentations nous ont montré la nécessité d'ajouter la contrainte d'ordre de suivi du texte. En effet, dans cette expérimentation aucun ordre n'était imposé et les étudiants négligaient la relecture et balayaient le texte avec la souris. Suite à ces mises à l'essai nous avons également intégré les contraintes de soulignage d'expressions et de lecture ou relecture complète présentées dans le chapitre 5, paragraphe 5.1.4.1, page 97.

Au cours de ces séances, nous avons observé les étudiants lors de la construction de leur diagramme. Nous avons en particulier remarqué que certains étudiants n'associaient pas de sémantique aux éléments du diagramme. Ces observations nous ont conduit à la conception des aides de création et reformulation des éléments graphiques.

6.1.2 Mise à l'essai des différents tâches d'apprentissage de la modélisation orientée objet

L'objectif de ces mise à l'essai était d'appliquer les différentes façons de constuire les diagrammes à compléter et à corriger que nous avons présentées dans le chapitre 4, paragraphe 4.2.1.1, page 70 et d'expérimenter ces tâches en environnement informatique pour valider les objectifs pédagogiques avancés. Pour ce faire, nous avons travaillé en collaboration avec Thierry Lemeunier. Nous nous somme basés sur les exercices de modélisation de type « Diagramme à créer » qu'il propose pour les étudiants de seconde année de DEUST ISR. Nous avons utilisé le diagramme de référence de chaque exercice pour élaborer les diagramme à corriger et à compléter conformément à nos propositions.

Protocole

Les mises à l'essai ont consisté en deux séances de trois heures de travaux pratiques avec des étudiants de DEUST ISR seconde année à l'université du Maine à l'automne 2007. Les étudiants ont travaillé à l'aide du logiciel commercial Objecteering [Objecteering], l'énoncé et le diagramme de départ étant fournis dans un format papier.

Nous avons testé à chaque séance deux exercices relevant de la tâche « Diagramme à compléter » et deux exercices de « Diagramme à corriger ».

Lors de la première séance, l'ébauche de diagramme à compléter ne comportait que les classes et l'apprenant devait ajouter les propriétés des classes et les relations. Concernant la seconde séance, nous avions proposé deux exercices dans lesquels des classes, attributs et relations avaient été supprimés en variant le niveau de difficulté.

Les exercices de diagramme à corriger de la première séance ne comportaient que des erreurs sur les relations. Pour cette séance, nous avions utilisé uniquement des erreurs de type erroné d'une relation (comme le remplacement d'une agrégation par une composition ou l'utilisation d'un héritage au lieu d'une association) ou des inversions de sens d'une relation. Les exercices de diagramme à corriger de la seconde séance comportaient

différents types d'erreur : représentation erronée d'une classe (remplacement par un attribut), transfert d'un attribut vers une autre classe, type erroné de relation, inversion du sens d'une relation. La résolution de cette tâche est complexe car il faut considérer l'ensemble du diagramme proposé et plus seulement les relations.

Observations

Concernant la tâche de « Diagramme à corriger », nous avions utilisé des énoncés d'exercices de création de diagrammes jugés difficiles. En donnant une ébauche de diagramme nous semblons réduire la difficulté de l'exercice, malgré le fait que cela nécessite en supplément une analyse du diagramme. Les étudiants ont dans l'ensemble obtenu des diagrammes similaires car ils partaient d'une base commune. Peu d'entre eux ont été perturbés par la modélisation qui leur était imposée par l'ébauche de diagramme.

Il nous paraît préférable pour ce type de tâche, de focaliser l'objectif pédagogique sur un élément précis, comme le choix du type de relation par exemple.

Les exercices à corriger sont apparus globalement plus difficiles que les exercices à compléter car il faut contrôler chaque élément du diagramme. Les consignes de correction locales (par exemple des modifications portant uniquement sur le type de relation) réduisent également la difficulté, mais peuvent permettre de travailler un aspect particulier de la modélisation orientée objet.

Afin de concevoir des exercices plus difficiles, il nous semble intéressant d'étudier la tâche de diagramme « à corriger et à compléter ». Le diagramme de départ pourrait à la fois comporter des erreurs de modélisation et être incomplet. Pour des exercices difficiles, cela contraindrait l'apprenant à une fine analyse du diagramme tout en veillant à la cohérence du diagramme final.

6.2 Expérimentations du modèle d'interaction

Deux séries d'expérimentations nous ont permis de valider nos propositions. Pour cela, nous avons expérimenté le système DIAGRAM dans sa globalité au cours de quatre séances. Nous avions plus particulièrement étudié l'effet des aides métacognitives au cours de séances antérieures.

Ces expérimentations se sont déroulées dans des conditions écologiques. Nous n'avons pas mis en place un protocole nécessitant un groupe de contrôle et un groupe test car cela semblait prématuré au regard du stade encore prototypique de développement de DIAGRAM. De plus, lors des expérimentations globales de DIAGRAM, le système de diagnostic était également en phase de test.

6.2.1 Evaluation globale du système

Nous avons mené à l'automne 2008 une campagne d'expérimentation de la version complète de DIAGRAM, intégrant les différents types d'activités, le module de diagnostic et les rétroactions.

6.2.1.1 Objectifs

Un premier objectif était d'expérimenter le système globalement afin de tester son utilité et son utilisabilité. Pour y parvenir, nous avons utilisé des questionnaires afin de recueillir l'opinion des apprenants sur leur utilisation des éléments intégrés à DIAGRAM

comme les outils, les aides ou les messages de rétroaction. Un second objectif était d'étudier plus précisément l'effet des messages de rétroaction afin de déterminer leur efficacité à permettre le contrôle et l'auto-correction de l'apprenant.

Lors de ces expérimentations, le système de diagnostic était également en phase de test, car il s'agissait de sa première expérimentation en situation réelle. Nous avons tenu compte dans notre analyse des conséquences des dysfonctionnements possibles de ce module sur la qualité des rétroactions.

6.2.1.2 Protocole

L'expérimentation s'est déroulée avec dix-huit étudiants de seconde année de DEUST ISR qui ont travaillé avec DIAGRAM durant quatre séances de trois heures. Elle a eu lieu en conditions écologiques, c'est-à-dire durant les séances de travaux pratiques intégrées au cursus des étudiants et en présence de l'enseignant. Les séances 1, 2 et 3 correspondaient aux premières séances de travaux pratiques de modélisation orientée objet des étudiants. Entre les séances 3 et 4, les apprenants ont travaillé durant deux autres TP avec Objecteering, un outil professionnel pour la modélisation [Objecteering]. Ces deux TP portaient sur la modélisation d'autres diagrammes étudiés durant le cours, comme par exemple les diagrammes d'états-transitions, et sur des diagrammes de classes intégrant des éléments qu'il n'est pas possible de créer dans DIAGRAM. Les exercices réalisés durant les séances 1, 2 et 3 étaient du type « diagramme à créer ». La séance 4 portait sur trois exercices : deux étaient de type « diagramme à créer », le dernier était du type « diagramme à compléter ».

Au début de l'expérimentation, les étudiants ont été avertis que les messages de rétroaction dans DIAGRAM étaient relatifs à une solution particulière et qu'ils pouvaient ignorer certains messages s'ils étaient convaicus que leur solution était correcte. L'enseignant leur a conseillé de lire les messages et d'évaluer si une modification était nécessaire. Chaque étudiant a réalisé trois à cinq exercices par séance, et nous avons enregistré l'intégralité des actions réalisées à l'interface . Un questionnaire a été distribué à la fin des séances 1, 2 et 4, afin de recueillir l'avis des étudiants sur DIAGRAM (cf. Annexe C, page 145).

6.2.1.3 Analyse des questionnaires

Les questionnaires distribués en fin de séances 1, 2 et 4 étaient identiques, à part une ou deux questions spécifiques (questions de prise en main pour les deux premières séances et comparaison avec Objecteering pour la dernière). Ils contenaient dix-huit assertions pouvant décrire un aspect de DIAGRAM. Les étudiants disposaient de quatre degrés d'accord pour donner leur impression (tout à fait d'accord, plutôt d'accord, plutôt pas d'accord, pas du tout d'accord) et d'une option « sans opinion ». Les résultats ont ensuite été regroupés en trois niveaux : d'accord, pas d'accord, et sans opinion. Les questionnaires contenaient également trois questions ouvertes permettant aux étudiants de s'exprimer sur les aspects positifs et négatifs de DIAGRAM et de proposer des suggestions d'amélioration.

A l'issue de la première séance, tous les participants (100 %) ont trouvé la prise en main de DIAGRAM facile et à l'issue de la seconde séance, 79 % ont déclaré avoir eu moins de difficulté que la première fois à utiliser DIAGRAM.

Quatre items du questionnaire concernaient l'aide apportée par les modes d'interaction et l'interface de DIAGRAM : le découpage en étapes, la présence du texte à l'écran, l'outil de soulignage et l'utilisation des couleurs. A l'issue de la séance 4, tous les étudiants

(100 %) sont d'accord avec le fait que le texte à l'interface et les couleurs facilitent la construction du diagramme. Pour le découpage en étapes et l'outil de soulignage, ce pourcentage augmente au fil des séances : pour le découpage en étapes, il passe de 85 % à 94 % et pour l'outil de soulignage, il passe de 69 % à 88 %.

Plusieurs items portaient sur les aides de création et de reformulation. 23 % (contre 54 %) des étudiants ont déclaré utiliser l'aide à la création, à l'issue de la première séance. Ce pourcentage passe à 65 % (contre 29 % de « pas d'accord ») à l'issue de la dernière séance. Dans le même temps, le pourcentage d'étudiants qui trouvent cette aide utile augmente également : il passe de 31 % (« pas d'accord » : 31 %) à 47 % (« pas d'accord » : 35 %). En moyenne, 61 % des étudiants déclarent avoir utilisé les aides de reformulation associées aux éléments graphiques. A l'issue de la première séance, seulement 38 % la jugeaient utile (« pas d'accord » : 62 %) mais ce pourcentage passe à 59 % (« pas d'accord » : 29 %) à l'issue de la quatrième séance.

Ces résultats suggèrent que les étudiants prennent conscience, à travers leur utilisation de DIAGRAM, de la nécessité d'anticiper et de contrôler leurs actions.

A l'issue de la dernière séance, une majorité d'étudiants (65 % contre 23 %) jugent les messages de rétroaction pertinents. Le pourcentage d'étudiants trouvant les trois modalités (indication, question, proposition) utiles passe de 54 % (« pas d'accord » : 46 %) à la première séance, à 76 % (« pas d'accord » : 18 %) après la dernière séance. A l'item « les messages de rétroaction m'ont permis de résoudre des problèmes de modélisation », les opinions des étudiants étaient partagées après la première séance (46 % « d'accord » contre 46 % « pas d'accord ») mais clairement positives à la fin des expérimentations (76 % « d'accord » contre 18 % « pas d'accord »). Le pourcentage d'étudiants trouvant que « l'utilisation de DIAGRAM aide dans à l'apprentissage des concepts de la modélisation orientée objet » est constant au fil des séances (86 % contre 12 %). En moyenne, 91 % des étudiants (contre 7 %) trouvent le travail avec DIAGRAM bénéfique pour l'apprentissage.

Le questionnaire distribué après la quatrième et dernière séance comportait deux items sur la comparaison entre DIAGRAM et Objecteering : 71 % des étudiants préfèrent travailler avec DIAGRAM et 23 % préférent travailler avec Objecteering.

Les questions ouvertes des questionnaires portaient sur les aspects positifs et négatifs de DIAGRAM et les suggestions d'amélioration. La plupart des réponses mettent en avant le fait que DIAGRAM est facile et rapide à utiliser, intuitif et insistent sur certains aspects (texte à l'interface, couleurs, aides) qui semblent particulièrement utiles aux étudiants. Parmi les faiblesses de DIAGRAM, sont mentionnés certaines difficultés avec l'interface (principalement le manque de raccourcis clavier), certains messages de rétroaction non pertinents et le découpage en étapes jugé trop contraignant. Les messages non pertinents peuvent s'expliquer par le fait que le module de diagnostic était également en phase de test et que certains appariements, sur lesquels s'appuient les rétroactions, étaient erronés. Les suggestions d'amélioration portent sur des aspects techniques comme l'ajout de raccourcis, de fonctions d'annulation et de répétition des actions ou l'amélioration de la sélection des objets graphiques.

Les résultats du questionnaire montrent que les apprenants novices ont globalement une opinion positive de DIAGRAM. On constate en particulier que le pourcentage d'opinion positive concernant les messages de rétroaction augmente après plusieurs séances d'utilisation. Ceci est probablement dû au fait que les étudiants arrivent à mieux comprendre l'utilité des messages après plusieurs séances. Bien que regrettant les limitations techniques dues à l'état de prototype du logiciel, les étudiants ont apprécié de travailler avec DIAGRAM et considèrent que les spécificités qu'il comporte facilitent

l'apprentissage de la modélisation orientée objet.

6.2.1.4 Evaluation des rétroactions

Nous avons analysé la plupart des exercices des deuxième et troisième séances. La première séance était une séance de prise en main de DIAGRAM et n'a pas été analysée. De plus, cette séance a été légèrement perturbée du fait que les étudiants n'ont pas compris certains messages de rétroaction liés à des concepts qui n'avaient pas encore été expliqués en cours, la progression du cours ayant été plus lente que prévue. Compte tenu de la difficulté des exercices et des différences importantes de rythme de travail entre les étudiants, les productions de la séance 4 ne sont pas assez significatives pour être analysées. De plus, cette séance comportait un exercice de type « diagramme à compléter », mais peu d'étudiants l'ont abordé.

Pour chaque exercice, nous avons compté le nombre d'appels au diagnostic, le nombre de messages et le nombre de messages lus. Nous considérons qu'un message est lu si l'apprenant ouvre au moins la première modalité du message (le plus souvent, l'indication). Nous avons analysé 86 exercices pour un total de 306 appels du diagnostic, soit en moyenne 3,56 appels par exercice et par étudiant. Les appels du diagnostic ont produit 1924 messages dont 836 (43,45 %) ont été lus. Certains étudiants adoptent la stratégie consistant à lire seulement un ou deux messages, modifier immédiatement le diagramme et rappeler ensuite le diagnostic, ce qui explique en partie le faible taux de lecture. Dans certains cas toutefois, les messages sont trop nombreux et les étudiants, probablement découragés, ne les lisent pas tous. Sur un exercice complet, si le nombre de messages est inférieur à 50, le taux de lecture est de 52,74 % alors que s'il est supérieur à 50, le taux chute à 31,25 %.

Un même message peut apparaître plusieurs fois au cours d'un exercice (au plus, autant de fois qu'il y a d'appels au diagnostic), si bien que les 836 messages correspondent à 565 messages distincts. Nous avons étudié les réactions des apprenants à la suite de ces 565 messages et distingué quatre cas :

- **Correction** : l'apprenant modifie son diagramme de manière cohérente avec le message (43,9 %) ;
- **Modification** : l'apprenant effectue une modification en lien avec le message mais son diagramme reste différent du diagramme de référence (3,4 %) ;
- **Sans effet** : l'apprenant ne modifie pas son diagramme (41 %) ;
- **Incohérent** : le module de diagnostic n'a pas correctement évalué les différences structurelles entre le diagramme de l'apprenant et le diagramme de référence. Les différences pédagogiques et le message de rétroaction qui en résulte sont donc incohérents (11,6 %).

Nous avons examiné plus précisément les diagnostics provoquant les messages de rétroaction incohérents et nous avons constaté qu'ils étaient dus à des appariements incorrects entre les éléments des deux diagrammes. Cela se produit lorsque le diagramme de l'apprenant est très différent du diagramme de référence, à la fois du point de vue de la structure et des noms des classes ou des relations, ou lorsque deux classes (ou plus) ont des noms voisins. Comme l'algorithme de diagnostic se base sur la structure et les noms pour comparer les diagrammes, ses performances se dégradent dans ce cas, ce qui avait déjà été constaté lors des premiers tests hors ligne de l'outil de diagnostic [Auxepaules et al., 2008]. Toutefois, cela n'a pas perturbé les séances car les incohérences étaient assez flagrantes et les étudiants ont généralement choisi d'ignorer ces messages.

Si l'on écarte les cas « incohérent », les cas restants se répartissent en 49,70 % de

« correction », 46,49 % de « sans effet » et 3,81 % de « modification ». Les cas de modification non corrective sont rares, ce qui suggère que les messages sont suffisamment clairs : lorsque les étudiants modifient leur diagramme pour prendre en compte un message, celui-ci est bien interprété (49,70 % des cas). En revanche, le taux de messages n'entraînant aucun effet est relativement élevé (46,49 %). Plusieurs hypothèses peuvent expliquer ce résultat. Comme nous avions expliqué aux étudiants qu'ils pouvaient ignorer certains messages, il est probable que beaucoup d'entre eux l'ont fait. De plus, nous avons observé plusieurs appels au diagnostic dans les dernières minutes de séances de TP. Dans ce cas, même si les apprenants ont lu les messages, ils n'ont pas eu le temps de réaliser toutes les modifications. Nous pouvons également supposer que certains messages n'ont pas été compris du tout, ou bien n'étaient pas assez précis pour que l'étudiant trouve par lui-même la correction à apporter. Afin de déterminer quels types de messages étaient concernés, nous avons analysé plus précisément le taux de correction en fonction du type de différence.

Le tableau 6.1 montre le pourcentage de chaque différence pédagogique parmi les messages lus et le taux de correction par différence. Les résultats montrent que les différences « Représentation erronée » et « Multiplicité erronée » sont les plus fréquentes (respectivement 28,46 % et 27,67 %). Les messages correspondants sont faciles à comprendre et précis en ce qui concerne la modification à effectuer. Le taux de correction montre que dans la moitié des cas les étudiants décident de réaliser la modification proposée. Les messages liés à la différence « Omission » sont généralement suggestifs et simples à suivre, particulièrement si le concept manquant dans le diagramme est présent dans l'énoncé : cela explique le bon taux de correction de cette différence. A l'inverse, le taux de correction de la différence « Ajout » est faible : comme l'ajout d'éléments aboutit souvent à une variante du diagramme de référence, les apprenants choisissent généralement de conserver les éléments ajoutés. La différence composée « Dédoublement et transfert » survient souvent lorsque les éléments communs d'une classe-mère sont doublés dans les classes-filles. Le fort taux de correction montre que les messages facilitent la compréhension du processus de généralisation, un concept important en modélisation orientée objet. Les messages liés aux différences « Dédoublement » et « Fusion » semblent trop généraux. Pour ce type de différence, il est difficile de produire un message qui soit à la fois générique et suffisamment explicite.

TAB. *6.1 – Pourcentage parmi les messages lus et taux de correction des différences pédagogiques*

Différence pédagogique	**% parmi les messages lus**	**Taux de correction**
Représentation erronée	28,46	51,41 %
Multiplicité erronée	27,65	50,72 %
Omission	15,83	67,09 %
Ajout	10,82	24,07 %
Dédoublement	8,42	21,43 %
Dédoublement et transfert (composée)	4,61	86,96 %
Fusion	2,81	21,43 %
Inversion de sens	1,20	100,00 %
Transfert	0,20	100,00 %
Total	100,00	

Nos avons observé que, dans certains cas, la lecture d'un message et la correction du diagramme conduisent l'apprenant à vérifier et à corriger d'autres éléments, proches de ceux concernés par le message. Cela suggère que les messages de rétroaction sollicitent efficacement la fonction métacognitive de vérification.

Globalement, le temps passé sur DIAGRAM après le premier diagnostic représente environ un tiers du temps total. Les étudiants ont donc consacré un temps important à vérifier et à modifier leur diagramme, en fonction des messages de rétroaction. Les diagrammes obtenus en fin de séance apparaissent dans l'ensemble plus complets et plus pertinents que les diagrammes tels qu'ils étaient avant le premier appel du diagnostic.

6.2.1.5 Conclusions

Les messages de rétroaction ont efficacement aidé les apprenants à réfléchir sur leur diagramme et à corriger leurs erreurs. Le taux de correction est moyen, conformément à nos prévisions, car les apprenants peuvent avoir imaginé des solutions alternatives et choisir d'ignorer les suggestions du logiciel. En revanche, le taux de lecture des messages est plus faible que nous l'avions imaginé. Il pourrait être amélioré par une meilleure organisation des messages. Par exemple, il serait possible de trier les messages par ordre d'importance, en fonction du type de différence, et de n'afficher que les premiers messages de la liste, afin d'attirer l'attention des étudiants sur les points importants (comme les omissions, les dédoublements et transferts), et de ne leur signaler les points mineurs (comme les multiplicités erronées) que dans un second temps.

6.2.2 Validation des aides métacognitives dans DIAGRAM

Nous avons réalisé deux campagnes d'expérimentations focalisées sur l'utilisation des aides métacognitives. La première s'est déroulée sur quatre séances à l'automne 2006, avec des étudiants de DEUST ISR de l'université du Maine. Nous avons eu l'opportunité de réaliser une seconde expérimentation en mars 2007. Elle consistait en une séance de travaux pratiques avec des étudiants de l'IUT Informatique de Laval. Disposer de ce public supplémentaire nous a permis de recueillir un nombre plus important de données. De plus, cela nous a permis d'expérimenter DIAGRAM avec un public différent.

6.2.2.1 Objectifs

Ces expérimentations avaient pour objectif de tester l'utilisation en contexte écologique des aides métacognitives de création et de reformulation par des apprenants. Concernant l'aide à la création, nous souhaitions vérifier l'utilisation de cette aide et son efficacité à soutenir le contrôle de son activité par l'apprenant. Pour l'aide de reformulation, nous avions émis l'hypothèse que confronter l'apprenant à la reformulation textuelle de son diagramme pouvait l'aider à valider ou invalider ses constructions. Nous souhaitions donc évaluer l'utilisation de cette aide par les apprenants, et son influence sur les aptitudes métacognitives de vérification et d'auto-correction.

6.2.2.2 Protocole

Ces expérimentations se sont toutes déroulées en conditions écologiques, c'est-à-dire durant des séances de travaux pratiques de modélisation orientée objet intégrées au cursus d'enseignement des étudiants. De plus, l'enseignant habituel de TP était présent afin de répondre aux questions et de suivre le déroulement des TP. Au cours des séances,

les étudiants utilisaient une version de DIAGRAM intégrant uniquement l'activité de « Diagramme à créer » selon la méthodologie en trois étapes. Le module de diagnostic et les rétroactions n'étaient pas encore implémentés. Nous avions utilisé un logiciel d'enregistrement des actions réalisées à l'écran pour recueillir les vidéos de toutes les séances.

La première expérimentation, réalisée à l'automne 2006, s'est déroulée sur quatre séances de TP de trois heures et concernait cinq étudiants de seconde année de DEUST ISR à l'université du Maine. Ceux-ci utilisaient déjà la méthode de modélisation en trois étapes en travaux dirigés. Nous avons récolté et analysé les vidéos des étudiants : ils ont réalisé entre 9 et 14 exercices chacun (sur 15 proposés) pour un total de 59 exercices.

La seconde expérimentation s'est déroulée en mars 2007. Elle impliquait 28 étudiants de première année de DUT (Diplôme Universitaire de Technologie) Informatique à l'IUT de Laval. Répartis en deux groupes, les étudiants ont tous travaillé durant 3h de TP. Contrairement aux étudiants de DEUST ISR, ces étudiants n'étaient pas habitués à l'utilisation de la méthode en trois étapes. 6 exercices ont été proposés aux étudiants qui en ont réalisé entre 5 et 6 pour un total de 150 exercices enregistrés et analysés.

6.2.2.3 Analyses des productions et résultats

Une première analyse des enregistrements nous a montré que l'aide à la création d'éléments est peu utilisée. Cela peut s'expliquer par le délai d'affichage des infobulles et la nécessité de l'immobilité du curseur. Les étudiants ont tendance à effectuer l'action trop rapidement, si bien que l'infobulle n'est pas affichée. Le dispositif de reformulation par contre est bien exploité par les apprenants pour contrôler leurs productions. Nous avons donc focalisé notre étude sur l'utilisation des infobulles de reformulation des relations. Pour cela, nous avons identifié et repéré les séquences d'infobulles concernant les relations, affichées au cours de la phase de construction du diagramme. Ces séquences sont constituées de l'affichage d'une infobulle suivi d'une ou plusieurs actions (modification, suppression de la relation) et éventuellement d'une nouvelle infobulle concernant la même relation. Nous avons mis en évidence quatre catégories de séquences :

- lorsque la relation reformulée est valide et que l'apprenant ne modifie pas la relation, il s'agit d'un cas de **vérification** ;
- si la reformulation permet à l'apprenant de détecter une erreur et de la corriger, c'est un cas de **correction** ;
- les séquences sont classées **sans effet** lorsque la relation est erronée mais que la reformulation ne permet pas à l'apprenant de corriger son erreur ;
- enfin, le cas de **dégradation** est celui pour lequel la reformulation conduit l'apprenant à modifier sa solution et à introduire une erreur alors que la relation est correcte.

Durant la série d'expérimentations de l'automne 2006, chaque étudiant travaillant avec DIAGRAM a réalisé entre 9 et 14 exercices pour un total de 307 séquences d'infobulles. Cela représente en moyenne 5,2 séquences par étudiant et par exercice. Durant l'expérimentation de mars 2007, chaque étudiant a réalisé entre 5 et 6 exercices pour un total de 284 séquences d'infobulles dans DIAGRAM. Cela représente en moyenne 1,89 séquences par étudiant et par exercice. Le tableau 6.2 présente la répartition des séquences d'infobulles sur les relations selon les quatre types de séquences. Dans les deux expérimentations, les infobulles sont principalement utilisées pour vérifier la correction de la relation.

TAB. *6.2 – Répartition des séquences d'infobulles*

	Expérimentation n°1	**Expérimentation n°2**	**Moyenne**
Vérification	50,5 %	62 %	56,2 %
Correction	18 %	12,3 %	15,2 %
Sans Effet	30,9 %	25 %	27,9 %
Dégradation	0,6 %	0,7 %	0,7 %

Nous avons ensuite affiné notre analyse pour nous intéresser aux cas des infobulles reformulant une relation qui comporte une erreur. Dans ces cas là, soit l'apprenant corrige son diagramme après l'affichage de l'infobulle (cas de *correction*), soit il n'effectue pas de modification (cas *sans effet*). Nous distinguons pour chaque cas les erreurs d'orientation des relations des autres erreurs (comme les erreurs liées au type de la relation, ou aux classes auxquelles la relation est liée). Comme nous l'avons expliqué dans le chapitre 4 paragraphe 4.1.3.2, page 67, les erreurs d'orientation sont particulièrement intéressantes car nous avons constaté que les novices éprouvent fréquemment des difficultés à percevoir la sémantique véhiculée par l'orientation des relations.

TAB. *6.3 – Répartition des séquences Correction et Sans Effet*

		Expérimentation n°1	**Expérimentation n°2**	**Moyenne**
Correction	Erreur d'orientation	56,4 %	62,9 %	59,7 %
	Autre erreur	43,6 %	37,1 %	40,3 %
Sans Effet	Erreur d'orientation	5,3 %	11,3 %	8,3 %
	Autre erreur	94,7 %	88,7 %	91,7 %

Les résultats présentés dans le tableau 6.3 montrent que les cas de *correction* sont répartis de façon équitable : une correction du diagramme suite à l'affichage d'une infobulle correspond dans 56,4 % des cas à la correction d'une erreur d'orientation, et dans 43,6 % des cas à la correction d'un autre type d'erreur. En revanche, nous constatons un fort déséquilibre pour les cas *sans effet* : plus de 90 % des séquences *sans effet* concernent une erreur autre que l'orientation.

Enfin, pour mettre en évidence l'influence du type d'erreur (erreur d'orientation ou autre erreur) sur la relation de l'apprenant, nous avons regroupé les résultats en fonction du type d'erreur (cf. tableau 6.4). Sur l'ensemble des expérimentations, les erreurs d'orientation sont corrigées dans 79,7 % des cas alors que les autres types d'erreur ne sont corrigées que dans 19,1 % des cas.

TAB. *6.4 – Répartition des séquences selon le type d'erreur*

		Expérimentation n°1	**Expérimentation n°2**	**Moyenne**
Erreur d'orientation	Correction	86,1 %	73,3 %	79,7 %
	Sans effet	13,9 %	26,7 %	20,3 %
Autre erreur	Correction	21,1 %	17,1 %	19,1 %
	Sans effet	78,9 %	82,9 %	80,9 %

Ces résultats peuvent s'expliquer par le fait que, lorsque l'orientation d'une relation est inversée, la phrase de reformulation apparaît incorrecte de manière flagrante. C'est en particulier le cas si c'est le type de la relation qui est reformulé (« Une figure géométrique est une sorte de cercle » au lieu de « Un cercle est une sorte de figure géométrique » par exemple). Pour d'autres erreurs, la phrase peut être sémantiquement correcte. L'erreur est alors moins évidente à repérer.

6.2.2.4 Conclusion

Nos analyses ont montré que l'aide à la création était peu utilisée. Nous n'avons donc pas pu démontrer son influence sur le contrôle de son activité par l'apprenant. En revanche, l'analyse de l'affichage des infobulles sur les relations montre que les infobulles sont principalement utilisées pour la *vérification*. Ces résultats peuvent s'expliquer par le fait que les apprenants ne prennent pas le temps d'afficher l'infobulle de création mais contrôlent rapidement la relation créée en affichant sa reformulation. Les infobulles facilitent également la correction des erreurs : l'affichage de la reformulation semble assez efficace pour les erreurs d'orientation (79,7 %). Le système est efficace à un moindre degré (19,1 %) pour les autres types d'erreur. Enfin, nous ne constatons pas d'effet négatif des infobulles (moins de 1 % des cas).

Les résultats de nos analyses montrent que le système de reformulation assure effectivement une fonction de *monitoring* au sens de la régulation métacognitive. Il permet également aux apprenants d'exercer l'aptitude d'auto-correction, en particulier pour les erreurs d'orientation des relations.

6.3 Synthèse

Ce chapitre présente les différentes expérimentations réalisées au cours de nos travaux de recherche. Elles ont d'une part permis la mise au point du modèle et d'autre part son expérimentation au sein d'un EIAH en situation écologique. De ces expérimentations, nous retenons les éléments suivants : l'aide de reformulation des éléments a montré une certaine efficacité, en particulier dans les cas où le message montre l'erreur de manière flagrante. Cela dénote aussi la limite de ces aides. La reformulation est locale et ne prend pas en compte les erreurs impliquant d'autres éléments du diagramme. Une évolution de cette aide serait de proposer une reformulation globale du diagramme. Concernant les rétroactions, elles ont efficacement aidé les apprenants à vérifier leur diagramme mais leur organisation pourrait être améliorée. Certains messages pourraient également être perfectionnés pour être plus pertinents. Enfin, les résultats demandent à être confirmés par des expérimentations plus poussées, avec un groupe de contrôle et un groupe de test. Cela nécessite un protocole complexe et notamment un nombre significatif d'apprenants sur plusieurs séquences de travail et un logiciel robuste implémentant des fonctionnalités d'utilisabilité poussées, ce qui n'est pas actuellement le cas de DIAGRAM.

CHAPITRE 7

Bilan des travaux et perspectives

Nous nous sommes intéressés dans ces travaux de recherche à l'interaction au sein d'un environnement informatique pour l'apprentissage de la modélisation orientée objet. Nos contributions sont la proposition d'un modèle d'interaction pour un environnement d'apprentissage de la modélisation et son application au domaine de la modélisation orientée objet avec UML.

Un modèle d'interaction pour l'apprentissage de la modélisation informatique

Nous nous sommes appliqués, dans nos travaux, à dégager un certain niveau de généricité. L'ensemble des composants du modèle décrits ci-dessous peut donc s'appliquer à tout domaine de la modélisation informatique. Le modèle comporte :

- un ensemble de tâches (modèle à créer, à compléter ou à corriger) pouvant s'appliquer dans différents domaines de la modélisation et ayant des objectifs pédagogiques variés.
- un ensemble d'outils (soulignage, surlignage, rattachement et passage de la souris) facilitant la modélisation dans l'environnement graphique et basés sur l'intégration de l'énoncé à l'interface de l'EIAH. Nous proposons également une scénarisation des activités d'apprentissage selon une méthode de modélisation en étapes ;
- des aides qui soutiennent l'activité métacognitive de l'apprenant au cours de la modélisation. Ces aides présentent un aspect générique car elles se basent principalement sur une reformulation de la sémantique des éléments graphiques de modélisation.
- des rétroactions qui soutiennent également l'activité métacognitive, notamment par une structuration en trois modalités que sont l'indication, la question et la proposition.

Dimension métacognitive du modèle d'interaction

Un des objectifs visés par ce modèle est de favoriser l'activité métacognitive de l'apprenant. L'intégration d'une dimension métacognitive dans nos propositions a guidé une partie importante de nos travaux. Cela se retrouve dans nos propositions. L'organisation en étapes favorise la planification. L'aide à la création et la reformulation des éléments soutiennent l'auto-correction et le contrôle de son activité par l'apprenant. Les rétroactions que nous proposons ont pour objectif de faire réfléchir l'apprenant à la pertinence de sa solution et encouragent la vérification et l'auto-contrôle. Les résultats obtenus lors des expérimentations sont encourageants et témoignent de l'intérêt d'une telle approche.

Limites

Nous nous sommes focalisés sur l'apprentissage de la modélisation en phase d'initiation et cela entraîne plusieurs limites à l'application de notre modèle au sein d'un

cadre plus général. Tout d'abord, si les activités structurées en étapes semblent constituer un moyen de cadrer le travail de l'apprenant en phase d'initiation, elles semblent difficilement applicables lors d'un enseignement plus approfondi. De même, le texte affiché à l'interface et l'utilisation de couleurs ne peuvent raisonnablement pas être appliqués au delà d'un certain seuil de lisibilité et de densité d'information à l'écran. Enfin, nous nous sommes limités aux éléments utiles en phase d'initiation. L'élaboration de rétroactions pertinentes prenant en compte de manière exhaustive tous les éléments d'un domaine serait une tâche complexe.

Discussion de la généricité du modèle

L'aspect de notre modèle dont la généricité prête le plus à discussion est la formulation des rétroactions. Le caractère générique des rétroactions réside dans la formulation en trois modalités et dans les objectifs des rétroactions. De plus, elles visent à soutenir l'activité métacognitive en favorisant l'auto-réflexion de l'apprenant. Toutefois, pour être efficaces, les rétroactions doivent être adaptées au domaine et aux productions de l'apprenant. Le contenu même des messages dépend donc fortement du domaine de modélisation dans lequel ces éléments sont mis en œuvre.

Cependant, le cheminement que nous avons suivi pour élaborer les rétroactions dans le cadre de l'apprentissage de la modélisation orientée objet peut être considéré comme l'ébauche d'une méthode : tout d'abord, nous avons travaillé en collaboration avec un expert et enseignant du domaine d'instanciation. Nous avons analysé des modèles réalisés par des apprenants lors de travaux pratiques afin de repérer les différences entre le modèle produit par l'apprenant et le modèle de référence, fourni par l'enseignant. A partir de ces éléments, nous avons élaboré une taxonomie de rétroactions et formulé les messages, selon les trois modalités, en mettant l'accent sur l'incitation à la vérification du diagramme par l'apprenant.

Perspectives

Il serait nécessaire d'éprouver la généricité de notre modèle d'interaction en l'appliquant à un autre domaine comme par exemple, la modélisation de schémas Entité-Relation dans les bases de données. Pour cela, nous pouvons réutiliser les tâches et activités génériques, la méthodologie en étapes et les outils de modélisation. Les aides métacognitives peuvent s'appliquer aux entités et aux relations. Les cardinalités des relations pourraient être reformulées sur le même principe que la reformulation des multiplicités.

Application du modèle d'interaction : implémentation et expérimentations dans un EIAH pour l'apprentissage de la modélisation orientée objet

L'instanciation de notre modèle a consisté en l'application de nos propositions au domaine de la modélisation orientée objet et du diagramme de classes en particulier. Ainsi, nous avons étudié comment les tâches de modélisation proposées se déclinent pour l'apprentissage de la modélisation par diagramme de classes et nous avons formulé des exemples concrets de messages d'aides.

Ces propositions ont été instanciées dans DIAGRAM, un EIAH pour l'apprentissage de la modélisation orientée objet développé dans le cadre du projet « Interaction et connaissances dans les EIAH pour la modélisation ». Cela a donné lieu à un important travail de développement informatique et a permis d'effectuer plusieurs expérimentations en conditions écologiques.

Apports

Parmi les précédents travaux en EIAH, nos travaux se rapprochent des recherches du groupe [ICTG], autour des environnements Collect-UML et KERMIT. Nos travaux se démarquent par une interaction générale plus élaborée au sein de l'environnement informatique d'apprentissage. Cela se manifeste par un ensemble d'outils de modélisation et une structuration en étapes guidant la modélisation. De plus, nos propositions n'induisent pas de contraintes sur l'énoncé ou sur le modèle de référence : cela permet une plus grande liberté dans l'élaboration des exercices que dans KERMIT ou Collect-UML qui nécessitent un énoncé très précis.

Enfin, nous proposons des aides pour que l'apprenant évalue sa solution au cours de la modélisation, et nous axons nos rétroactions sur un soutien à l'activité métacognitive. Cette approche laisse plus de liberté à l'apprenant que KERMIT et Collect-UML qui interviennent dès qu'une contrainte est violée. Nos propositions se rapprochent des messages élaborés par [Tholander et al., 1999]. Notre avantage est d'utiliser une évaluation du diagramme après la modélisation et une phase de rétroactions séparée, ce qui permet de mieux gérer le moment d'intervention du système, principal écueil du système de [Tholander et al., 1999].

Eléments de validation et limites

Nous avons mené deux expérimentations dont les résultats nous donnent des éléments de validation de notre modèle. Elles montrent que la reformulation des éléments aide les apprenants à contrôler leur diagramme. C'est en particulier le cas pour les relations orientées. De même, les rétroactions ont aidé les apprenants à vérifier leur diagramme.

Une limite observée dans le cadre de l'implémentation de notre modèle dans DIAGRAM est la dépendance de la mise en œuvre des rétroactions à l'outil de diagnostic du système. En effet, bien que notre taxonomie de différences pédagogiques, sur laquelle se basent les rétroactions, ait été élaborée séparément, la réalisation informatique nécessite de passer par un outil de comparaison entre la production de l'apprenant et le modèle de référence. L'outil de diagnostic existant dans DIAGRAM utilise un seul diagramme de référence. Si l'apprenant produit un diagramme correct mais très différent structurellement du diagramme de référence, les rétroactions seront inadaptées. Toutefois, ce cas de figure nous semble peu probable en phase d'initiation.

Perspectives

Concernant nos travaux dans le cadre de l'apprentissage de la modélisation orientée objet, plusieurs perspectives de recherche sont envisagées.

A court terme, elles concernent principalement les rétroactions. L'objectif des rétroactions était d'encourager la vérification et l'auto-correction en posant notamment des questions à l'apprenant. Les questions sont principalement des questions fermées dont la réponse est soit positive, soit négative. On pourrait offrir la possibilité à l'apprenant de

répondre en ajoutant des boutons (par exemple : « oui », « non », « je ne sais pas », « je ne comprends pas la question »). En fonction de sa réponse, une nouvelle question ou une rétroaction plus pertinente et plus ciblée pourrait être proposée.

A long terme, une des perspectives du module de diagnostic est de prendre en compte plusieurs diagrammes de référence et de repérer celui dont le modèle construit par l'apprenant est le plus proche. Le système pourrait alors utiliser ce diagramme pour construire des rétroactions plus pertinentes.

On pourrait également envisager d'utiliser les versions successives du diagramme élaboré par l'apprenant en mémorisant les diagnostics au fur et à mesure. Cela permettrait d'éviter de répéter le même message lorsque, par exemple, le système détermine qu'une différence qu'il a repérée est un choix de modélisation de l'apprenant. Dans le cas contraire, cela permettrait également d'insister si une erreur persiste au cours d'un exercice. De manière plus générale, l'intégration d'un modèle de l'apprenant dans le système permettrait une meilleure adaptation à l'apprenant. Par exemple, cela permettrait de fournir des rétroactions plus globales en fonction des conceptions que le système repérerait chez l'apprenant.

Enfin, des expérimentations plus poussées seraient essentielles pour compléter la validation de nos propositions. Cela nécessiterait l'utilisation d'un nombre significatif d'apprenants, répartis en un groupe de contrôle et un groupe de test sur plusieurs séances de travail. Un tel protocole permettrait d'évaluer plus précisément les effets du modèle sur l'apprentissage.

Annexes

Liste des annexes

ANNEXE A

Différences pédagogiques et des rétroactions associées

Cette annexe présente les différences pédagogiques que nous avons identifiées, et les rétroactions qui y sont associées. Pour des raisons de lisibilité, nous nous limitons à décrire celles qui impliquent les éléments suivants : classe, attribut, opération, association, agrégation, composition, héritage.

Les différences simples sont décrites par catégories : ajout, omission, représentation erronée, transfert, inversion du sens de relation, dédoublement d'élément et fusion et multiplicités erronées. Nous donnons également quelques exemples de rétroactions associées à des différences pédagogiques composées, lorsqu'elles diffèrent de celles de la différence simple.

A.1 Différences pédagogiques simples

Ajout

Ajout d'une classe

Indiquer : La classe C n'est pas reliée au diagramme.
Questionner : La classe C est-elle utile ?

Ajout d'un attribut

Indiquer : Ta classe C possède l'attribut A.
Questionner : Est-ce que A est une propriété de C ?

Ajout d'une opération

Indiquer : Ta classe C peut réaliser l'opération O.
Questionner : Est-ce que O est un comportement de C ?

Ajout d'une relation

Indiquer : *selon le type de la relation*
– *association* : Tu dis que C1 est associé(e) à C2.
– *agrégation* : Tu dis que C1 est un ensemble de C2.
– *composition* : Tu dis que C1 est composé(e) de C2.
– *héritage* : Tu dis que C1 est associé(e) à C2.
Questionner : *selon le type de la relation*
– *association* : Est ce que C1 est associé(e) à C2 ?
– *agrégation* : Est ce que C1 est un ensemble de C2 ?
– *composition* :Est ce que C1 est composé(e) de C2 ?
– *héritage* : Est ce que C1 est associé(e) à C2 ?

Omission

Omission d'une classe

Cette différence ne peut pas se produire de manière isolée

Omission d'un attribut

Indiquer : Il manque une propriété dans la classe C.
Questionner : As-tu représenté toutes les propriétés de C ?
Proposer : Un(e) C est caractérisé(e) par un attribut A.

Omission d'une opération

Indiquer : Il manque une opération dans la classe C.
Questionner : As-tu représenté tous les comportements de C ?
Proposer : Un(e) C peut réaliser l'opération O.

Omission d'une relation

Indiquer : Il manque une relation entre les classes C1 et C2.
Questionner :
– *Si aucune relation n'est présente entre C1 et C2* : Quelle relation existe entre C1 et C2 ?
– *S'il existe déjà une ou plusieurs relations entre C1 et C2* : Quelle autre relation existe entre C1 et C2 ?
Proposer : *selon le type de la relation*
– *association* : C1 est associé(e) à C2.
– *agrégation* : C1 est un ensemble de C2.
– *composition* : C1 est composé(e) de C2.
– *héritage* : C1 est une sorte de C2.

Représentation erronée

Type erroné de relation

Association au lieu de Agrégation

Indiquer : Tu dis que C1 est associé(e) à C2.
Questionner : Est-ce que la relation R entre C1 et C2 n'est pas plus forte ?

Association au lieu de Composition

Indiquer : Tu dis que C1 est un ensemble de C2.
Questionner : Est-ce que la relation R entre C1 et C2 n'est pas plus forte ?

Agrégation au lieu de Composition

Indiquer : Tu dis que C1 est un ensemble de C2.
Questionner :
– Est-ce que la relation R entre C1 et C2 n'est pas plus forte ?

– Est-ce qu'un objet C2 peut être partageable entre plusieurs objets C1 ?

Agrégation au lieu de Association

Indiquer : Tu dis que C1 est un ensemble de C2.
Questionner :
– Est-ce qu'un objet C2 est une partie d'un objet C1 ?
– Est-ce qu'un objet C1 est un ensemble d'objets C2 ?

Composition au lieu de Association

Indiquer : Tu dis que C1 est composé(e) de C2.
Questionner :
– Est-ce qu'un objet C2 est une partie d'un objet C1 ?
– Est-ce qu'un objet C1 est composé d'objets C2 ?

Composition au lieu de Agrégation

Indiquer : Tu dis que C1 est composé(e) de C2.
Questionner :
– Est-ce qu'un objet C2 est une partie d'un objet C1 ?
– Est-ce qu'un objet C1 est composé d'objets C1 ?
– Est-ce qu'un objet C1 partage C2 avec d'autres classes ?

Association, Agrégation ou Composition au lieu de Héritage

Indiquer : *selon le type de la relation*
– *association* : C1 est associé(e) à C2.
– *agrégation* : C1 est un ensemble de C2.
– *composition* : C1 est composé(e) de C2.
Questionner : *selon le type de la relation*
– *association* : Est-ce qu'une des classes n'est pas plus générale/spécialisée que l'autre ?
– *agrégation* : Est-ce qu'une des classes n'est pas plus générale/spécialisée que l'autre ?
– *composition* : Est-ce que C1 et C2 ont des propriétés et/ou des comportements communs ?
Proposer : Je dirais plutôt que C1 est une sorte de C2.

Héritage au lieu de Association, Agrégation ou Composition

Indiquer : Tu dis que C1 est une sorte de C2.
Questionner : Est-ce que C1 est une sorte de C2 ?
Proposer :
– *association* : Je dirais plutôt que C1 est associé(e) à C2.
– *agrégation* : Je dirais plutôt que C1 est un ensemble de C2.
– *composition* : Je dirais plutôt que C1 est composé(e) de C2.

Classe au lieu d'attribut

Indiquer : Tu as représenté C comme une classe.
Questionner : La classe C a-t-elle des propriétés ?
Proposer : Si un(e) C est une propriété, c'est plutôt l'attribut d'une classe.

Transfert

Transfert d'attribut

Indiquer : Tu dis que A est une propriété de C1.
Questionner : Est-ce que A est une propriété C1 ?
Dans le cas d'un transfert vers une classe fille ou une classe mère C2, de C1
Questionner : Est-ce que l'attribut A est une propriété commune à toutes les sous-classes de C2 ?
Proposer : Je dirais plutôt qu'un(e) C2 est caractérisé(e) par un attribut A.

Transfert d'opération

Indiquer : Tu dis que O est un comportement de C1.
Questionner : Est-ce que O est un comportement de C1 ?
Dans le cas d'un transfert vers une classe fille ou une classe mère C2, de C1
Questionner : Est-ce que l'opération O est un comportement commun à toutes les sous-classes de C2 ?
Proposer : Je dirais plutôt qu'un(e) C2 peut réaliser l'opération O.

Transfert de relation

Indiquer : *selon le type de la relation*
– *association* : Tu dis que C1 est associé(e) à C2.
– *agrégation* : Tu dis que C1 est un ensemble de C2.
– *composition* : Tu dis que C1 est composé(e) de C2.
– *héritage* : Tu dis que C1 est associé(e) à C2.
Questionner : *selon le type de la relation*
– *association* : Est ce que C1 est associé(e) à C2 ?
– *agrégation* : Est ce que C1 est un ensemble de C2 ?
– *composition* :Est ce que C1 est composé(e) de C2 ?
– *héritage* : Est ce que C1 est associé(e) à C2 ?
Si les 2 classes C3 et C4 sont identifiées dans le diagramme
Proposer :
– *association* : Je dirais plutôt que C3 est associé(e) à C4.
– *agrégation* : Je dirais plutôt que C3 est un ensemble de C4.
– *composition* : Je dirais plutôt que C3 est composé(e) de C4.
– *héritage* : Je dirais plutôt que C3 est une sorte de C4.

Inversion du sens de relation

Indiquer : *selon le type de la relation*
– *agrégation* : Tu dis que C1 est un ensemble de C2.

– *composition* : Tu dis que C1 est composé(e) de C2.
– *héritage* : Tu dis que C1 est associé(e) à C2.

Questionner : *selon le type de la relation*
– *agrégation* : Est ce que C1 est un ensemble de C2 ?
– *composition* :Est ce que C1 est composé(e) de C2 ?
– *héritage* : Est ce que C1 est associé(e) à C2 ?

Proposer :
– *agrégation* : Je dirais plutôt que c'est C2 qui est un ensemble de C1.
– *composition* : Je dirais plutôt que c'est C2 qui est composé(e) de C1.
– *héritage* : Je dirais plutôt que c'est C2 qui est une sorte de C1.

Dédoublement

Dédoublement de classes

Questionner : Les classes C1…Cn représentent-elles le même concept ?
Proposer : Tu dois n'en garder qu'une seule.

Dédoublement d'attributs

Questionner : Les attributs A1…An représentent-ils la même propriété ?
Proposer : Tu dois n'en garder qu'un seul.

Dédoublement d'opérations

Questionner : Les opérations O1…On représentent-elles le même comportement ?
Proposer : Tu dois n'en garder qu'une seule.

Dédoublement de relations

Questionner : Les relations R1…Rn représentent-elles la même relation ?
Proposer : Tu dois les regrouper en une seule.

Fusion

Fusion de classes

Indiquer : Tu as représenté une classe C.
Proposer : N concepts différents doivent être représentés par n classes différentes.

Fusion d'attributs

Indiquer : Tu dis que la classe C possède un attribut A.
Proposer : Une classe a autant d'attributs que de propriétés.

Fusion d'opérations

Indiquer : Tu dis que la classe C peut réaliser le comportement O.
Proposer : Une classe a autant d'opérations que de comportements différents.

Fusion de relations

Indiquer : Tu as représenté une seule relation entre les classes C1 et C2.
Proposer : Il y a autant de relations entre les classes que de relations entre les concepts.

Multiplicité erronée

Nous ne décrirons qu'un cas de multiplicité erronée. Il s'agit du cas où la multiplicité X de l'apprenant se trouve entre les bornes [YBasse ...YHaute] de la multiplicité indiquée dans le diagramme de référence.

Multiplicité X pendant [YBasse ...YHaute]

Indiquer : Tu dis qu'un objet C1 est lié à exactement exactement X objets C2.
Questionner :
– Est-ce qu'un objet C1 est toujours lié à exactement X objets C2 ?
– Est-ce qu'un objet C1 ne peut pas être lié à moins de X objets C2 ?
– Est-ce qu'un objet C1 ne peut pas être lié à plus de X objets C2 ?
Proposer : Un objet C1 est plutôt lié à entre Ybasse et YHaute objets C2.

A.2 Différences composées

Différence principale : Ajout de classe

Différences associées éventuelles : Ajout d'attributs ou d'opérations dans cette classe, Dédoublement d'attributs ou d'opérations de cette classe, Dédoublement et transfert d'attributs ou d'opérations de cette classe, Ajout de relation sur cette classe, Transfert de relations vers cette classe, Dédoublement et transfert de relations vers cette classe, etc.
Indiquer : Tu représentes une classe C.
Questionner : Est-ce que la classe C est utile ?

Différence principale : Omission de classe

Différences associées éventuelles : Omission d'attributs ou d'opérations de cette classe, Transferts d'attributs ou d'opérations depuis cette classe, Dédoublement et transfert d'attributs ou d'opérations depuis cette classe, Omission de relations sur cette classe, Transfert de relations depuis cette classe, Dédoublement et transfert de relations depuis cette classe, etc.
Indiquer : Le diagramme est incomplet. Un concept important n'est pas représenté.
Questionner : As-tu représenté tous les concepts du diagramme ?

Attribut au lieu de classe

Indiquer : Tu as représenté A comme un attribut de la classe C.
Questionner : A peut-il avoir des propriétés et/ou des comportements ?
Proposer : Si A possède des propriétés et des comportements, c'est plutôt une classe.

Dédoublement et transfert d'attributs

Indiquer : Tu dis que C1 est caractérisé par A1,...Cn est caractérisé par An.
Questionner :
- Les attributs A1 de la classe C1,...An de la classe Cn, représentent-ils la même propriété ?
- Dans un héritage, où sont représentées les propriétés communes aux sous-classes ?
- Dans un héritage, qu'est-ce qui est hérité par les sous-classes ?

Proposer :
- *si la classe C est identifiée dans le diagramme de l'apprenant* : Tu dois regrouper ces attributs en un seul, en utilisant la classe C.
- *si la classe C n'est pas identifiée dans le diagramme de l'apprenant* : Tu dois regrouper ces attributs en un seul.

Dédoublement et transfert d'opérations

Indiquer : Tu dis que C1 peut réaliser l'opération O1,...Cn peut réaliser l'opération On.
Questionner :
- Les opérations O1 de la classe C1, ...On de la classe Cn, représentent-elles le même comportement ?
- Dans un héritage, où sont représentées les opérations communes aux sous-classes ?
- Dans un héritage, qu'est-ce qui est hérité par les sous-classes ?

Proposer :
- *si la classe C est identifiée dans le diagramme de l'apprenant* : Tu dois regrouper ces opérations en une seule, en utilisant la classe C.
- *si la classe C n'est pas identifiée dans le diagramme de l'apprenant* : Tu dois regrouper ces opérations en une seule.

ANNEXE B

Evaluation métrique du développement réalisé dans DIAGRAM

Dans cette annexe, nous présentons l'évaluation métrique du développement effectué dans DIAGRAM. Nous avons utilisé pour cela un *plugin* Eclipse appelé Metrics [Metrics]. Cet outil permet d'obtenir 23 données métriques d'un projet comme le nombre de lignes de code, de classes, d'attributs, de packages, la profondeur du code, etc. Nous avons retenu trois métriques permettant de quantifier le code de DIAGRAM : le nombre total de lignes de code (sans ligne blanche ni commentaire), le nombre de packages et le nombre de classes.

Afin d'évaluer précisément nos contributions, nous avons comparé les données métriques de chaque package des versions préliminaire et finale de DIAGRAM présentées dans le chapitre 5, paragraphe 5.2.1, page 105.

La version préliminaire de DIAGRAM contient 5 modules qui sont le module de gestion de l'application, le composant de gestion des données source, l'éditeur textuel, l'éditeur graphique et le module de gestion du scénario. L'application comporte également trois modules utilitaires que nous regroupons dans notre analyse. Elle ne contient pas les modules de rétroactions et de diagnostic. Le module de scénario est réduit à une version figée des étapes de lecture, modélisation et relecture du scénario de diagramme à créer. Le tableau B.1 détaille le nombre de lignes de code, de package et de classes par module de cette version.

TAB. *B.1 – Evaluation métrique de la version préliminaire de* DIAGRAM

Module	**Nombre de lignes de code**	**Nombre de packages**	**Nombre de classes**
Gestion de l'application	592	2	18
Gestion des données source	546	1	7
Editeur textuel	760	3	14
Editeur graphique	3501	9	54
Scénario	115	1	4
Utilitaire	231	2	2
Total	**5745**	**18**	**99**

La version « finale » de DIAGRAM contient en plus le module des rétroactions et le composant de diagnostic. Le tableau B.2 détaille le nombre de lignes de code, de package et de classes par module de cette version.

Ces données métriques montrent que le module de diagnostic représente 70 % des lignes de code de la version finale de DIAGRAM. En mettant de coté ce module, l'application contient 13012 lignes de code alors que la version préliminaire en comportait 5745. Pour la mise en œuvre de nos propositions nous avons implémenté le module des rétroactions qui représente 6 % des lignes de code et 22 % des packages de l'application finale. De plus, nous avons modifié l'application préliminaire : l'implémentation des différents scénarios dans DIAGRAM a multiplié par 9 le nombre de lignes de code de ce

TAB. *B.2 – Evaluation métrique de la version finale de* DIAGRAM

Module	**Nombre de lignes de code**	**Nombre de packages**	**Nombre de classes**
Gestion de l'application	1479	2	24
Gestion des données source	823	1	8
Editeur textuel	1838	3	15
Editeur graphique	4774	9	58
Scénario	1046	1	20
Rétroactions	2676	11	48
Diagnostic	31338	19	69
Utilitaire	376	4	4
Total	**44350**	**50**	**246**

module. Nous avons multiplié par 2,5 le nombre de lignes de code du module de gestion de l'application (afin d'implémenter en particulier le chargement et la sauvegarde du travail de l'apprenant). Le nombre de lignes de code de l'éditeur textuel a été multiplié par 2,4 pour l'implémentation des différents outils de modélisation. L'éditeur graphique contient 1,3 fois plus de lignes de code. Les principaux ajouts concernent l'implémentation des aides métacognitives et des outils de modélisation.

ANNEXE C

Questionnaires distribués lors des expérimentations globales de DIAGRAM

Cette annexe présente les résultats des questionnaires distribués lors des expérimentations de DIAGRAM réalisées à l'automne 2008. Trois questionnaires ont été distribués, à la fin des séances 1, 2 et 4. Sur 18 étudiants, 13 ont répondu au premier questionnaire, 14 au deuxième et 17 au dernier.

Préambule

Tous les questionnaires comportaient le préambule suivant :

Vous allez lire un certain nombre de phrases qui peuvent décrire vos impressions durant votre utilisation de DIAGRAM, comme :

DIAGRAM est facile à utiliser
☐ Tout à fait d'accord
☐ Plutôt d'accord
☐ Plutôt pas d'accord
☐ Pas du tout d'accord

☐ Sans opinion

Si vous pensez que DIAGRAM n'est pas du tout facile à prendre en main, cochez la case « Pas du tout d'accord ». Dans ce cas, vous n'êtes pas du tout d'accord avec cette affirmation. A l'opposé, si vous pensez que DIAGRAM est très facile à prendre en main, cochez la case « Tout à fait d'accord ». Dans ce cas, vous êtes complètement d'accord avec cette assertion.

Vous pouvez utiliser quatre degrés d'accord (ou de désaccord) proposés. Si une phrase n'a aucun sens pour vous ou si vous pensez qu'elle ne s'applique pas vraiment à DIAGRAM, choisissez la case « Sans opinion ».

Vous pouvez aussi faire des commentaires.

Nous avons regroupé les réponses « Tout à fait d'accord » et « Plutôt d'accord » en « D'accord », les réponses « Plutôt pas d'accord » et « Pas du tout d'accord » en « Pas d'accord » et les « Sans opinion » et « Pas de réponse » en « Sans opinion ».

Prise en main

DIAGRAM est facile à utiliser, la prise en main est rapide

	Expé. n°1
D'accord	100 %
Pas d'accord	0 %

J'ai eu moins de difficultés à utiliser DIAGRAM que la première fois

	Expé. n°2
D'accord	79 %
Pas d'accord	21 %

Modes d'interaction et interface

Le découpage en étapes aide à réaliser le diagramme correctement

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	85 %	93 %	94 %
Pas d'accord	15 %	7 %	6 %

La présence du texte à l'écran facilite la construction du diagramme

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	100 %	100 %	100 %
Pas d'accord	0 %	0 %	0 %

L'utilisation d'un outil de soulignage de l'énoncé dans la première étape facilite la construction du diagramme

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	69 %	86 %	88 %
Pas d'accord	31 %	7 %	12 %
Sans opinion	0 %	7 %	0 %

L'utilisation de couleurs pour les mots de l'énoncé et les éléments du diagramme facilite la construction du diagramme

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	92 %	93 %	100 %
Pas d'accord	8 %	7 %	0 %

Aides du logiciel DIAGRAM

J'ai utilisé l'aide à la création

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	23 %	57 %	65 %
Pas d'accord	54 %	43 %	29 %
Sans opinion	23 %	0 %	6 %

J'ai trouvé appropriés (pertinents) les messages d'aide à la création

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	31 %	64 %	59 %
Pas d'accord	23 %	36 %	29 %
Sans opinion	46 %	0 %	12 %

L'aide à la création m'a permis d'éviter des erreurs de construction du diagramme

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	31 %	36 %	47 %
Pas d'accord	31 %	57 %	35 %
Sans opinion	38 %	7 %	18 %

J'ai utilisé les reformulations associées aux éléments graphiques du diagramme

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	69 %	50 %	65 %
Pas d'accord	23 %	36 %	29 %
Sans opinion	8 %	14 %	6 %

J'ai trouvé appropriés (pertinents) les messages reformulant les éléments graphiques

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	54 %	71 %	65 %
Pas d'accord	31 %	22 %	29 %
Sans opinion	15 %	7 %	6 %

Les messages de reformulation m'ont permis de résoudre des erreurs de construction du diagramme

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	38 %	57 %	59 %
Pas d'accord	62 %	36 %	29 %
Sans opinion	0 %	7 %	12 %

Messages de rétroaction proposés par DIAGRAM

J'ai trouvé appropriés (pertinents) les messages de rétroaction

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	69 %	47 %	65 %
Pas d'accord	8 %	36 %	23 %
Sans opinion	23 %	21 %	12 %

L'organisation globale des messages de rétroaction (éléments à corriger / éléments ajoutés / éléments omis) m'a aidé dans le processus de validation du diagramme

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	70 %	64 %	71 %
Pas d'accord	15 %	29 %	23 %
Sans opinion	15 %	7 %	6 %

Les différentes modalités des messages de rétroaction (Indication / Question / Proposition) m'ont aidé dans le processus de validation du diagramme

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	54 %	71 %	76 %
Pas d'accord	46 %	22 %	18 %
Sans opinion	0 %	7 %	6 %

J'ai systématiquement regardé toutes les modalités associées à un message de rétroaction

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	15 %	50 %	47 %
Pas d'accord	85 %	43 %	47 %
Sans opinion	0 %	7 %	6 %

Les messages de rétroaction m'ont permis de résoudre des problèmes de modélisation de mon diagramme

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	46 %	79 %	76 %
Pas d'accord	46 %	14 %	18 %
Sans opinion	8 %	7 %	6 %

Evaluation générale

Globalement, le travail avec DIAGRAM m'a aidé dans mon apprentissage des concepts de la modélisation orientée objet

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	85 %	86 %	88%
Pas d'accord	15 %	14 %	6 %
Sans opinion	0 %	0 %	6 %

J'ai le sentiment que le travail avec ce logiciel m'a été profitable

	Expé. n°1	*Expé. n°2*	*Expé. n°4*
D'accord	92 %	93 %	88 %
Pas d'accord	8 %	7 %	6 %
Sans opinion	0 %	0 %	6 %

DIAGRAM *vs* Objecteering (pour les diagrammes de classes)

J'ai préféré travailler avec DIAGRAM qu'avec Objecteering

	Expé. n°4
D'accord	71 %
Pas d'accord	23 %
Sans opinion	6 %

J'ai trouvé le travail avec DIAGRAM plus enrichissant que sous Objecteering

	Expé. n°4
D'accord	53 %
Pas d'accord	18 %
Sans opinion	29 %

Bibliographie

[Allal et Saada-Robert, 1992]
Linda Allal, Madelon Saada-Robert. *La métacognition : cadre conceptuel pour l'étude des régulations en situation scolaire*. In Archives de psychologie, 60(235) : pages 265–296, 1992. Cité page 46.

[Alphonce et Martin, 2005]
Carl Alphonce, Blake Martin. *Green : a pedagogically customizable round-tripping UML class diagram Eclipse plug-in*. In Proceedings of the Eclipse technology eXchange workshop, Satellite Event of the 20th Annual ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'05), pages 115–119. ACM, 2005. Cité page 29.

[Alphonce et Ventura, 2003]
Carl Alphonce, Phil Ventura. *QuickUML : a tool to support iterative design and code development*. In Proceedings of the 18th Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'03), pages 80–81. ACM, 2003. Cité page 29.

[Amado Gama, 2004]
Claudia Amado Gama. *Integrating Metacognition Instruction in Interactive Learning Environment*. Thèse de Doctorat, University of Sussex, 2004. Cité pages 45, 48, 50, 51, 53, 55, 57, 59 et 69.

[Antony et Batra, 2002]
Solomon R. Antony, Dinesh Batra. *CODASYS : a consulting tool for novice database designers*. In SIGMIS Database, 33(3) : pages 54–68, 2002. Cité page 33.

[Auxepaules et al., 2007]
Ludovic Auxepaules, Dominique Py, Thierry Lemeunier. *Une méthode de diagnostic reposant sur l'appariement structurel de diagrammes de classes dans un EIAH pour la modélisation orientée objet*. In Actes de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH'07), pages 527–532. 2007. Cité pages 93 et 94.

[Auxepaules et al., 2008]
Ludovic Auxepaules, Dominique Py, Thierry Lemeunier. *A diagnostic method that matches class diagrams in a learning environment for object-oriented modeling*. In Proceedings of the 8th International Conference on Advanced Learning Technologies (ICALT'08), pages 26–30. 2008. Cité pages 112 et 120.

[Auxepaules, 2008]
Ludovic Auxepaules. *Une méthode de diagnostic basée sur l'appariement de modèles dans un EIAH pour la modélisation orientée objet*. In Actes des secondes Rencontres Jeunes Chercheurs en EIAH (RJC EIAH'08), pages 79–84. 2008. Cité page 94.

[Baghaei et al., 2006]
Nilufar Baghaei, Antonija Mitrovic, Warwick Irwin. *Problem-Solving Support in a*

Constraint-based Intelligent Tutoring System for UML. In Technology, Instruction, Cognition and Learning (TICL), 4(1-2), 2006. Cité page 38.

[Baghaei et Mitrovic, 2006]
Nilufar Baghaei, Antonija Mitrovic. *A Constraint-based Collaborative Environment for Learning UML Class Diagrams*. In Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS'06), pages 176–186. 2006. Cité pages 38 et 58.

[Baghaei, 2007]
Nilufar Baghaei. *A collaborative constraint-based intelligent system for learning object-oriented analysis and Design using UML*. Thèse de Doctorat, University of Canterbury, New Zealand, 2007. Cité pages 38, 39 et 40.

[Bézivin et Gerbé, 2001]
Jean Bézivin, Olivier Gerbé. *Towards a Precise Definition of the OMG/MDA Framework*. In Proceedings of the 16th IEEE/ACM International Conference on Automated Software Engineering (ASE'01), page 273. 2001. Cité page 16.

[Brown, 1978]
Ann Leslie Brown. *Knowing when, where and how to remember : a problem of metacognition*. In Advances in Instructional Psychology, pages 77–165. R. Glaser, éditeur, volume 1. Lawrence Erlbaum Associates, Hillsdale, New Jersey, USA, 1978. Cité pages 46 et 47.

[Charroux et al., 2005]
Benoît Charroux, Aomar Osmani, Yann Thierry-Mieg. *UML 2*. Pearson Education France, 2005. Cité pages 25 et 26.

[Chi et al., 1989]
Micheline T.H. Chi, Miriam Bassok, Matthew W. Lewis, Peter Reimann, Robert Glaser. *Self-Explanation : How Students Study and Use Examples in Learning to Solve Problems*. In Cognitive Science, 13 : pages 145–182, 1989. Cité page 49.

[Conati et VanLehn, 2000]
Cristina Conati, Kurt VanLehn. *Toward Computer-Based Support of Meta-Cognitive Skills : a Computational Framework to Coach Self-Explanation*. In International Journal of Artificial Intelligence in Education (IJAIED), 11 : pages 389–415, 2000. Cité pages 53 et 54.

[Constantino-González et Suthers, 2000]
María de los Angeles Constantino-González, Daniel D. Suthers. *A Coached Collaborative Learning Environment for Entity-Relationship Modeling*. In Proceedings of the 5th International Conference on Intelligent Tutoring Systems (ITS'00), pages 324–333. Springer-Verlag, 2000. Cité page 34.

[Crahen et al., 2002]
Eric Crahen, Carl Alphonce, Phil Ventura. *QuickUML : a beginner's UML Tool*. In Proceedings of 17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'02), pages 62–63. ACM, 2002. Cité page 29.

[Dranidis, 2007]
Dimitris Dranidis. *Evaluation of StudentUML : an Educational Tool for Consistent Modelling with UML*. In Proceedings of the 2nd Informatics Education Europe Conference (IEEII), pages 248–256. 2007. Cité page 32.

[Favre et Estublier, 2006]
Jean-Marie Favre, Jacky Estublier. *Concepts de base de l'IDM – Modèle, métamodèle, transformation, mégamodèle*. In L'ingénierie dirigée par les modèles – au delà du MDA. Hermès, 2006. Cité page 16.

[Flavell, 1976]
John H. Flavell. *Metacognitive aspects of problem solving*. In The nature of intelligence, pages 231–235, 1976. Cité page 46.

[Flavell, 1979]
John H. Flavell. *Metacognition and cognitive monitoring : a new area of cognitive-developmental inquiry*. In American Psychologist, 34 : pages 906–911, 1979. Cité pages 47, 48 et 51.

[Flavell, 1981]
John H. Flavell. *Cognitive Monitoring*. In Children's oral communication skills, pages 35–59, 1981. Cité page 46.

[Frosch-Wilke, 2003]
Dirk Frosch-Wilke. *Using UML in Software Requirements Analysis – Experiences from Pratical Student Project Work*. In Informing Science inSITE, pages 175–183. 2003. Cité page 28.

[Gayler et al., 2007]
Dick Gayler, David Klappholz, Valerie J. Harvey, Manuel A. Pérez-Quiñones. *UML tools : what is their role in undergraduate computer science courses ?* In Proceedings of the 38th SIGCSE technical symposium on Computer science education (SIGCSE'07), pages 129–130. ACM, 2007. Cité page 28.

[Ghandeharizadeh, 2003]
Shahram Ghandeharizadeh. *An Educational Perspective on Database Managment Systems and Object-Oriented Methodology : A 12 Year Journey*. In Proceedings of the 18th Annual ACM Conference on Object Oriented Programming, Systems, Languages, and Applications (OOPSLA'03), pages 137–139. 2003. Cité page 28.

[Goodman et al., 1997]
Bradley Goodman, Amy Soller, Frank Linton, Robert Gaimari. *Encouraging Student Reflection and Articulation using a learning Companion*. In Proceedings of the 3rd International Conference on Artificial Intelligence in Education (AIED'97), pages 151–158. IOS Press, 1997. Cité page 58.

[Habra et Noben, 2001]
Naji Habra, Karl Noben. *Teaching Object Orientation at the Design Level*. In Proceedings of the 5th Workshop on Pedagogies and Tools for Assimilating Object-Oriented Concepts, 16th Annual ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'01). ACM, 2001. Cité page 27.

[Hall et Gordon, 1998]
Lynne Hall, Adrian Gordon. *A virtual learning environment for entity relationship modelling*. In SIGCSE Bulletin, 30(1) : pages 345–349, 1998. Cité pages 28 et 33.

[Hansen et Ratzer, 2002]
Klaus Marius Hansen, Anne Vinter Ratzer. *Tool Support for Collaborative Teaching and Learning of Object-Oriented Modeling*. In Proceedings of the 7th

Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'02), pages 146–150. ACM, 2002. Cité page 32.

[Hartman, 2001]
Hope J. Hartman. *Developing Students' Metacognitive Knowledge and Skills*. In Metacognition in Learning and instruction. Theory, Research and Practice, pages 33–68. Kluwer academic publishers, 2001. Cité page 49.

[Holland et al., 1997]
Simon Holland, Robert Griffiths, Mark Woodman. *Avoiding Objet Misconceptions*. In Proceedings of the 28th SIGCSE technical symposium on Computer science education (SIGCSE'97), pages 131–134. ACM, 1997. Cité page 28.

[Kölling et al., 2003]
Michael Kölling, Bruce Quig, Andrew Patterson, John Rosenberg. *The BlueJ system and its pedagogy*. In Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology, 13(4), 2003. Cité page 29.

[Komis et al., 2003]
Vassilis Komis, Nikolaos Avouris, Angélique Dimitracopoulou, Meletis Margaritis. *Aspects de la conception d'un environnement collaboratif de modélisation à distance*. In Actes de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH'03), pages 271–282. C. Desmoulins, P. Marquet, D. Bouhineau, éditeurs. 2003. Cité page 49.

[Laborde et Laborde, 2006]
Colette Laborde, Jean-Marie Laborde. *Genèse et développement de Cabri-géomètre, environnement de géométrie dynamique*. In Environnements Informatiques pour l'Apprentissage Humain, pages 345–374. M. Grandbastien, J.-M. Labat, éditeurs, Traité IC2 Information Commande Communication. Hermès, 2006. Cité page 67.

[Lemeunier, 2000]
Thierry Lemeunier. *L'intentionnalité communicative dans le dialogue homme-machine en langue naturelle*. Thèse de Doctorat, Université du Maine, 2000. Cité page 69.

[Mackay et Fayard, 1997]
Wendy E. Mackay, Anne-Laure Fayard. *HCI, Nature Science and Design : A framework for Triangulation across Disciplines*. In Designing Interactive Systems. 1997. Cité page 11.

[Meyer et Wendehals, 2004]
Matthias Meyer, Lothar Wendehals. *Teaching Object-Oriented Concepts with Eclipse*. In Proceedings of the Eclipse Technology eXchange Workshop, Satellite Event of the 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'04), pages 1–5. ACM Press, 2004. Cité page 29.

[Michel, 2006]
Johan Michel. *Modèles d'activités pédagogiques et de support à l'interaction pour l'apprentissage d'une langue*. Thèse de Doctorat, Université du Maine, 2006. Cité page 68.

[Moritz et al., 2005]
Sally H. Moritz, Fang Wei, Shahida M. Parvez, Glenn D. Blank. *From objects-first*

to design-first with multimedia and intelligent tutoring. In Proceedings of the 10th Annual conference on Innovation and technology in Computer Science Education (ITiCSE'05), pages 99 – 103. 2005. Cité page 36.

[Moritz et Blank, 2005]
Sally H. Moritz, Glenn D. Blank. *A Design-First Curriculum for Teaching Java in a CS1 Course*. In ACM SIGCSE Bulletin archive, 37(2) : pages 89–93, 2005. Cité page 37.

[Moritz, 2008]
Sally H. Moritz. *Generating and Evaluating Object-Oriented Designs in an Intelligent Tutoring System*. Thèse de Doctorat, Lehigh University, 2008. Cité page 36.

[Muller et Gaertner, 2000]
Pierre-Alain Muller, Nathalie Gaertner. *Modélisation objet avec UML*. Eyrolles, 2ème édition, 2000. Cité page 26.

[Palacio-Quintin, 1990]
Ercilia Palacio-Quintin. *L'éducation cognitive à l'école*. In European Journal of Psychology of Education, 2 : pages 231–242, 1990. Cité page 48.

[Parvez, 2007]
Shahida M. Parvez. *A pedagogical framework for integrating individual learning style into an ITS*. Thèse de Doctorat, Lehigh University, 2007. Cité page 38.

[Penders, 2002]
Tom Penders. *Introduction à UML*. OEM, 2002. Cité pages 25, 26 et 28.

[Puntambekar et du Boulay, 1999]
Sadhana Puntambekar, Benedict du Boulay. *Design of MIST – A System to Help Students Develop Metacognition*. In Learners, Learning Assement, pages 245–257. SAGE, 1999. Cité page 58.

[Py et al., 2008]
Dominique Py, Mathilde Alonso, Ludovic Auxepaules, Thierry Lemeunier. *Conception de rétroactions dans un EIAH pour la modélisation orientée objet*. In Actes du colloque Technologies de l'Information et de la Communication pour l'Enseignement (TICE'08), pages 27–34. 2008. Cité page 93.

[Ramollari et Dranidis, 2007]
Ervin Ramollari, Dimitris Dranidis. *StudentUML : an Educational Tool Supporting Object-Oriented Analysis and Design*. In Proceedings of the 11th Panhellenic Conference on Informatics (PCI'07), pages 363–373. 2007. Cité page 32.

[Rey-Debove et Rey, 2002]
Josette Rey-Debove, Alain Rey. *Le Petit Robert de la langue Française*. Editions Le Robert, 3ème édition, 2002. Cité page 16.

[Robbins, 1999]
Jason Elliot Robbins. *Cognitive Support Features for Software Development Tools*. Thèse de Doctorat, University of California, Irvine, 1999. Cité page 31.

[Roll et al., 2006]
Ido Roll, Eunjeong Ryu, Jonathan Sewall, Brett Leber, Bruce McLaren, Vincent Aleven, Kenneth R. Koedinger. *Towards Teaching Metacognition : Supporting Spontaneous Self-Assessment*. In Proceedings of the 8th International Conference

on Intelligent Tutoring Systems (ITS'06), pages 738–740. 2006. Cité pages 55 et 56.

[Romero, 2003]
Margarida Romero. *Apprentissage dans un EIAH à l'aide des outils de modélisation de connaissances*, LIUM, Université du Maine, Mars 2003. Rapport de Master 1. Cité page 58.

[Romero, 2004]
Margarida Romero. *Métacognition dans les EIAH*, LIUM, Université du Maine, Juin 2004. Exposé transversal. Cité pages 48 et 50.

[Romero, 2005]
Margarida Romero. *10 principes pour la prise en compte du développement métacognitif dans les EIAH*, LIUM, Université du Maine, Avril 2005. Cité pages 45, 50, 52, 53 et 59.

[Roques et Vallée, 2007]
Pascal Roques, Frank Vallée. *UML 2 en action, De l'analyse des besoins à la conception*. Eyrolles, 4ème édition, 2007. Cité pages 25 et 26.

[Roques, 2003]
Pascal Roques. *UML par la pratique*. Eyrolles, 2ème édition, 2003. Cité pages 25 et 26.

[Sakdavong et al., 2008a]
Jean-Christophe Sakdavong, Françoise Adreit, Nathalie Huet, Fabrice Noury. *Conception d'aides au guidage métacognitif de l'apprenant : Le dispositif logiciel ACMAMA*. In Actes du colloque Technologies de l'Information et de la Communication pour l'Enseignement (TICE'08), pages 144–146. 2008. Cité page 57.

[Sakdavong et al., 2008b]
Jean-Christophe Sakdavong, Nathalie Huet, Fabrice Noury, Françoise Adreit. *Méthodologie de conception et d'aides au guidage métacognitif dans l'apprentissage d'un traitement de texte*. In Journée d'Etude sur le Traitement Cognitif des Systèmes d'Information Complexes (JETSIC'08). 2008. Cité page 57.

[Suraweera et Mitrovic, 2004]
Pramuditha Suraweera, Antonija Mitrovic. *An intelligent Tutoring System for Entity Relationship Modelling*. In International Journal of Artificial Intelligence in Education (IJAIED), 14(3-4) : pages 375–417, 2004. Cité page 35.

[Surcin et al., 1995]
Sylvain Surcin, Christine Choppy, Marie-Geneviève Séré. *Analyse didactique d'un cours de Génie Logiciel basé sur l'approche orientée objets*. In Sciences et Techniques Educatives (STE), 2(3) : pages 265–286, 1995. Cité page 28.

[Tchounikine, 2002]
Pierre Tchounikine. *Pour une ingénierie des Environnements Informatiques pour l'Apprentissage Humain*. In Revue I3 information – interaction – intelligence, 2(1), 2002. Cité page 9.

[Tenbergen et al., 2008]
Bastian Tenbergen, Colleen Grieshaber, Lisa Lazzaro, Rick Buck. *SketchUML : A*

Tablet PC-based e-Learning Tool for UML syntax using a Minimalistic Interface. In Proceedings of the IADIS International Conference Mobile Learning, pages 84–91. 2008. Cité page 32.

[Tholander et al., 1999]
Jakob Tholander, Klas Karlgren, Fredrik Rutz, Robert Ramberg. *Design and Evaluation of an Apprenticeship Settinh for Learning Object-Oriented Modeling*. In Proceedings of the 7th International Conference on Computers in Education (ICCE'99), pages 399–406. 1999. Cité pages 15, 36, 40, 41, 42, 50, 58, 69, 129 et 161.

[Turner et al., 2005]
Scott A. Turner, Manuel A. Pérez-Quiñones, Stephen H. Edwards. *minimUML : A minimalist approach to UML diagramming for early computer science education*. In Journal on Educational Resources in Computing (JERIC), 5(4), 2005. Cité page 30.

[VanLehn, 1996]
Kurt VanLehn. *Conceptual and meta learning during coached problem solving*. In Proceedings of the 3rd International Conference on Intelligent Tutoring Systems (ITS'96), pages 29–47. 1996. Cité page 53.

[Weerasinghe et Mitrovic, 2002]
Amali Weerasinghe, Antonija Mitrovic. *Enhancing learning through self-explanation*. In Proceedings of the 10th International Conference on Computers in Education (ICCE'02), pages 244–248. 2002. Cité pages 36, 54 et 55.

[Weerasinghe et Mitrovic, 2006]
Amali Weerasinghe, Antonija Mitrovic. *Faciliting Deep Learning Through Self-Explanation in an Open-ended Domain*. In International Journal of Knowledge-Based and Intelligent Engineering Systems, pages 3–19. IOS Press, 2006. Cité pages 36 et 54.

[Wei, 2007]
Fang Wei. *A student model for an Intelligent Tutoring System helping novices learn Object Oriented Design*. Thèse de Doctorat, Lehigh University, 2007. Cité page 37.

[Zakharov et al., 2005]
Konstantin Zakharov, Antonija Mitrovic, Stellan Ohlsson. *Feedback Micro-engineering in EER-Tutor*. In Proceedings of the 12th International Conference on Artificial Intelligence in Education (AIED'05), pages 718–725. IOS Press, 2005. Cité page 36.

[Zimmerman, 2001]
Barry J. Zimmerman. *Theories of Self-Regulated Learning and Academic Achievement : An Overview and Analysis*. In Self-Regulated Learning and Academic Achievement, 2 édition, pages 1–38. B. J. Zimmerman, D. H. Schunk, éditeurs. Lawrence Erlbaum Associates, 2001. Cité page 47.

Webographie

[AgileJ] Site de AgileJ. Dernière consultation : 5 mars 2009.
url : http://www.agilej.com/. Cité page 29.

[ArgoUML] Site de ArgoUML. Dernière consultation : 5 mars 2009.
url : http://argouml.tigris.org/. Cité page 31.

[Astade] Site de ASTADE. Dernière consultation : 5 mars 2009.
url : http://astade.tigris.org/. Cité page 29.

[Bouml] Site de BOUML. Dernière consultation : 5 mars 2009.
url : http://bouml.free.fr/. Cité page 29.

[Fujaba] Site du projet FUJABA, University of Paderborn, Software Engineering Group. Dernière consultation : 5 mars 2009.
url : http://wwwcs.upb.de/cs/fujaba/projects/eclipse/index.html. Cité page 29.

[Gaphor] Site de Gaphor. Dernière consultation : 5 mars 2009.
url : http://gaphor.devjavu.com/. Cité page 29.

[Green] Site de Green. Dernière consultation : 5 mars 2009.
url : http://green.sourceforge.net/. Cité page 29.

[ICTG] Site du projet ICTG (Intelligent Computer Tutoring Group). University of Canterbury, New Zeland.
url : http://ictg.canterbury.ac.nz/. Cité pages 36, 38 et 129.

[IdeogramicUML] Site de Ideogramic UML. Dernière consultation : 5 mars 2009.
url : http://www.ideogramic.com/products/uml/. Cité page 32.

[Javahelp] Site officiel de Javahelp. Dernière consultation : 2007.
url : http://java.sun.com/javase/technologies/desktop/javahelp/. Cité page 106.

[Jude] Site de JUDE. Dernière consultation : 5 mars 2009.
url : http://jude.change-vision.com/jude-web/index.html. Cité page 29.

[MagicDraw] Site de MagicDraw. Dernière consultation : 5 mars 2009.
url : http://www.nomagic.com/dispatcher.php. Cité page 29.

[Metrics] Site de Metrics. Dernière consultation : 5 octobre 2009.
url : http://metrics.sourceforge.net/. Cité page 143.

[MinimUML] Site de MinimUML. Dernière consultation : 5 mars 2009.
url : http://minimuml.cs.vt.edu/. Cité page 30.

[Objecteering] Site de Objecteering. Dernière consultation : 5 mars 2009.
url : http://www.objecteering.com/products_uml_modeler.php. Cité pages 29, 116 et 118.

[Omondo] Site de Omondo. Dernière consultation : 5 mars 2009.
url : http://www.omondo.com/. Cité page 29.

[Papyrus] Site de Papyrus. Dernière consultation : 5 mars 2009.
url : http://www.papyrusuml.org. Cité page 29.

[Poseidon] Site de Poseidon for UML (GentleWare). Dernière consultation : 5 mars 2009.
url : http://www.gentleware.com/products.html. Cité page 29.

[RationalRose] Site de Rational Rose. Dernière consultation : 5 mars 2009.
url : http://www-01.ibm.com/software/awdtools/developer/rose/modeler/. Cité page 29.

[SketchUML] Site de SketchUML. Dernière consultation : 13 mars 2009.
url : http://sketchuml.tenbergen.org/. Cité page 32.

[Together] Site de Together (Borland). Dernière consultation : 5 mars 2009.
url : http://www.borland.com/us/products/together/. Cité page 29.

[Umbrello] Site de Umbrello. Dernière consultation : 5 mars 2009.
url : http://uml.sourceforge.net/. Cité page 29.

[UMLet] Site de UMLet. Dernière consultation : 5 mars 2009.
url : http://www.umlet.com/. Cité page 30.

[UMLStudio] Site de UMLStudio, Pragsoft corporation. Dernière consultation : 5 mars 2009.
url : http://www.pragsoft.com/. Cité page 29.

[Violet] Site de Violet. Dernière consultation : 5 mars 2009.
url : http://horstmann.com/violet/. Cité page 30.

[VisualParadigm] Site de Visual Paradigm. Dernière consultation : 5 mars 2009.
url : http://www.visual-paradigm.com/product/vpuml/. Cité page 29.

Table des figures

Liste des tableaux