

DIAGRAM, un EIAH pour l'initiation à la modélisation orientée objet avec les diagrammes de classe UML

Mathilde Alonso, Ludovic Auxepaules, Dominique Py

▶ **To cite this version:**

Mathilde Alonso, Ludovic Auxepaules, Dominique Py. DIAGRAM, un EIAH pour l'initiation à la modélisation orientée objet avec les diagrammes de classe UML. STICEF (Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation), 2010, 17, 22 p. hal-00696313

HAL Id: hal-00696313
https://hal.science/hal-00696313

Submitted on 11 May 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Sciences et Technologies de l´Information et de la Communication pour l´Éducation et la Formation

Volume 17, 2010
Article de recherche

DIAGRAM, un EIAH pour l'initiation à la modélisation orientée objet avec les diagrammes de classe UML

Mathilde ALONSO, Ludovic AUXEPAULES, Dominique PY
(Laboratoire d'Informatique de l'Université du Maine, Le Mans)

■ RÉSUMÉ : Cet article présente Diagram, un ˆ IAH pour la modélisation orientée objet avec les diagrammes de classe UML. Diagram réifie un modèle d'interaction conçu pour favoriser l'activité de régulation métacognitive chez l'apprenant. Ce modèle d'interaction repose sur une organisation de la tâche en plusieurs étapes, l'intégration de l'énoncé dans l'interface, des outils de modélisation graphique spécifiques et des aides contextuelles pour la création et la vérification des éléments du diagramme. Diagram intègre un module de diagnostic qui compare le diagramme de l'apprenant à un diagramme de référence et produit la liste des différences entre ces diagrammes. L'algorithme de diagnostic s'inspire des méthodes d'appariement de graphes et exploite des motifs structurels qui orientent l'appariement des diagrammes à comparer. À partir des différences repérées dans le diagramme de l'étudiant, des rétroactions sont élaborées. Nous illustrons le processus de production des rétroactions en donnant un exemple complet de résultats du diagnostic et de messages générés par Diagram. ˆ nfin, nous décrivons une expérimentation menée en contexte écologique et analysons les effets des rétroactions.

■ MOTS CLÉS : modélisation, UML, diagramme de classe, interaction, rétroaction, diagnostic, métacognition.

■ ABSTRACT : This paper presents Diagram, a learning environment for object-oriented modelling with UML class diagrams. Diagram reifies an interaction model that supports the learner's metacognitive activity. This model relies on a task organization, on specific modelling graphic tools and on contextual helps. Diagram includes a diagnostic module which compares the student diagram with a reference diagram, and produces the list of the differences between these diagrams. The diagnostic algorithm is inspired from graph comparison methods, and uses structural patterns that direct the matching of the diagrams to be compared. The differences that are noticed in the student diagram give rise to feedbacks. We illustrate the whole feedback elaboration process by giving an example of a student diagram with the diagnostic results and the messages generated by the learning environment. We describe an experimentation in ecological context and analyze the effects of the feedbacks.

■ Kˆ YWORDS : modelling, UML, class diagram, interaction, feedback, diagnosis, metacognition.

- 1. Introduction
- 2. L'apprentissage de la modélisation orientée objet
- 3. Cadre général de Diagram
- 4. L'interaction dans Diagram
- 5. Analyse du diagramme de l'apprenant
- 6. Rétroactions pédagogiques
- 7. ˆ xpérimentation
- 8. Conclusion

1. Introduction

ˆ n génie logiciel, la modélisation est une étape essentielle du processus de développement. ˆ lle est désormais enseignée dans les cursus informatiques universitaires à finalité professionnelle, surtout depuis l'avènement du langage UML (*Unified Modelling Language*). La conception d'ˆ IAH dédiés à la

modélisation a fait l'objet de recherches récentes, notamment sur le langage UML (Baghaei et al., 2006) (Moritz, 2008) ou sur les bases de données (Suraweera et Mitrovic, 2004). Dans ces EIAH, la pertinence des rétroactions pédagogiques repose sur la capacité du système à évaluer les productions de l'apprenant. La difficulté provient du fait que, dans une tâche ouverte comme la modélisation, il n'existe pas de méthode formelle pour construire une solution ni pour vérifier la validité d'un modèle. Cet obstacle conduit souvent les concepteurs à imposer des contraintes sur l'interaction, ce qui limite les possibilités d'expression de l'élève et le champ d'application de l'EIAH.

Dans le cadre d'un projet du Laboratoire d'Informatique de l'Université du Maine, nous avons exploré une autre approche visant à surmonter ces limites et proposer un environnement d'apprentissage plus ouvert. Cette approche repose sur un modèle d'interaction spécifique et sur une méthode de diagnostic qui évalue le modèle construit par l'apprenant et permet de produire des rétroactions pédagogiques tenant compte de la validité du diagramme de l'apprenant.

Nous décrivons en section 2 le contexte de ce travail et présentons brièvement l'état de l'art. La section 3 précise les objectifs visés et l'approche retenue dans Diagram. Après avoir décrit le modèle d'interaction en section 4, nous présentons la méthode de diagnostic en section 5 et les rétroactions pédagogiques en section 6. Enfin, la section 7 décrit une expérimentation du logiciel et présente ses résultats.

2. L'apprentissage de la modélisation orientée objet

Avec l'arrivée des langages dédiés à la modélisation orientée objet, les travaux autour de l'enseignement de ce domaine se multiplient : des études empiriques sont menées sur l'apprentissage de la modélisation, des ouvrages et des outils pédagogiques spécifiques sont développés. En nous appuyant sur une brève revue de ces travaux (un panorama plus large est disponible dans (Alonso, 2009) et (Auxepaules, 2009)), nous présentons dans cette section le contexte général du projet et précisons les objectifs et les limites de notre étude.

2.1. Le langage UML

UML (*Unified Modelling Language*) est un langage standardisé en 1997 par l'OMG (*Object Modelling Group*) (UML, 1997) qui facilite, indépendamment de tout langage de programmation, la conception de programmes, ainsi que leur description pour des non-informaticiens. Bien que UML soit issu de travaux sur les méthodes d'analyse et de conception orientée objet, 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 l'approche RUP (*Rational Unified Process*), mais ils ne font pas partie du standard UML. UML est donc un langage qui peut être utilisé quelle que soit la méthode choisie. Il a fait l'objet de nombreux ouvrages à visée pédagogique, tels que (Roques, 2003), (Charroux et al., 2005), (Roques et Vallée, 2007).

UML est fondé sur un métamodèle qui décrit formellement la syntaxe, la sémantique et la notation visuelle de chaque élément utilisé dans les diagrammes, ainsi que les relations entre ces éléments. La version actuelle d'UML 2.1 est constituée de treize types de diagrammes, représentant autant de points de vue d'un système logiciel. Parmi ces diagrammes, le diagramme de classes est au centre du processus de modélisation. Il est utilisé dans les phases d'analyse et de conception et peut représenter les informations avec différents niveaux d'abstraction et de précision en fonction de leur utilisation et de leur degré d'affinement. C'est le diagramme le plus employé et le mieux connu. De ce fait, il est souvent le premier type de diagramme enseigné dans les cours de modélisation orientée objet. C'est pourquoi nous avons choisi de cibler notre étude sur le diagramme de classes et ses principaux éléments : les classes, les relations et les attributs.

2.2. 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, qui correspondent à différents points de vue. Par ailleurs, UML ne fournissant pas de méthodologie de modélisation, il n'existe pas de démarche standard pour élaborer un diagramme de classes.

Les difficultés spécifiques à l'apprentissage de la modélisation orientée objet ont fait l'objet d'études

empiriques (Moisan et Rigault 2009). Ces études 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 ne peuvent s'acquérir que par la pratique. Ainsi, (Surcin et al., 1995) identifie la notion d'abstraction comme une notion fondamentale du cours de génie logiciel, mais la considère comme difficile à enseigner. (Habra et Noben, 2001) soulignent qu'aucune règle universelle ne s'applique pour la création d'un modèle, et notent que les apprenants ont besoin de conseils méthodologiques pour guider leur créativité. Ce point de vue est partagé par (Frosch-Wilke, 2003) qui estime que la modélisation s'acquiert par la pratique et recommande de fournir aux apprenants des processus ou des étapes à suivre pour construire leur diagramme.

Ces constats sur les difficultés rencontrées par les débutants nous ont conduits à étudier plus précisément quels seraient les modes d'organisation de l'interaction permettant de leur offrir des conditions d'apprentissage adaptées. Nous avons choisi de cibler notre travail en nous orientant vers un environnement d'apprentissage destiné spécifiquement aux novices, qui doit fournir des aides, notamment méthodologiques, pour aider les débutants à acquérir une première expérience de modélisation sur des exercices simples. En revanche, nous n'avons pas abordé dans ce projet l'apprentissage de notions plus complexes, ni la modélisation de projets de taille « réaliste », tels qu'ils sont abordés au niveau master notamment.

2.3. Métacognition et apprentissage

Le terme de métacognition, évoqué à l'origine dans (Flavell, 1976), désigne la connaissance qu'un sujet possède de ses propres processus cognitifs. Cette notion de métacognition se décline en deux dimensions : les connaissances métacognitives d'un individu, qui reflètent la prise de conscience de ses propres connaissances, et les régulations métacognitives, qui se réfèrent aux activités supportant le contrôle individuel de la pensée ou de l'apprentissage. C'est cette seconde dimension de régulation qui nous intéresse ici. Pour (Brown, 1987), la régulation métacognitive comprend trois fonctions principales : la planification (*planning*) d'activités à entreprendre, le contrôle (*monitoring*) d'activités en cours et la vérification (*checking*) des résultats d'activité.

Flavell considère que la métacognition joue un rôle important dans de nombreux domaines, dont l'apprentissage (Flavell, 1979). Selon l'approche constructiviste, l'apprenant 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. Les activités de régulation renforcent le contrôle de ses choix par l'apprenant et lui permettent d'avoir un regard critique sur son travail.

Nous avons montré que la modélisation est un processus ouvert, faisant appel à des savoirs et à des compétences acquises par l'expérience, et qui requiert des capacités d'abstraction. Il nous semble donc important de favoriser les capacités métacognitives de l'apprenant et notamment celles qui mettent en jeu la régulation : la planification, le contrôle et l'évaluation des résultats de son activité. Nous faisons ainsi l'hypothèse qu'un EIAH supportant une tâche dans laquelle l'activité métacognitive est essentielle doit prendre en compte et encourager ces trois dimensions de la régulation.

2.4. Environnements d'apprentissage pour les diagrammes de classe

Des produits gratuits aux systèmes haut de gamme, il existe aujourd'hui des dizaines d'éditeurs UML, dont certains possèdent des fonctionnalités très étendues. Ces outils sont adaptés à une utilisation professionnelle, mais de nombreuses fonctionnalités sont inutiles, voire perturbantes, pour une initiation à la modélisation orientée objet.

En parallèle, des outils éducatifs pour l'apprentissage du paradigme orienté objet ont été développés. Ces outils consistent généralement en une interface simplifiée, contenant un sous-ensemble d'éléments UML jugés utiles pour une introduction à la modélisation, ainsi que des fonctionnalités graphiques permettant de créer et de manipuler aisément les diagrammes. Certains logiciels, comme ArgoUML (Robbins, 1999) ou StudentUML (Dranidis, 2007), offrent également des conseils à l'apprenant pour vérifier et améliorer son diagramme. Ces conseils sont génériques et ne prennent pas en compte la validité du diagramme de l'apprenant.

Dans la recherche en EIAH, les travaux consacrés à l'apprentissage de la modélisation avec UML sont moins nombreux. Deux approches principales ont été explorées. La première, illustrée par

l'environnement Collect-UML (Baghaei et al., 2006), est basée sur la notion de contraintes. La seconde, utilisée dans DesignFirst-ITS (Moritz, 2008), repose sur un module expert pour analyser la solution de l'apprenant.

La modélisation à base de contraintes (*Constraint-Based Modelling*), proposée par Ohlsson (Ohlsson, 1994), consiste à définir les règles du domaine sous forme de contraintes, puis à vérifier que la réponse de l'apprenant satisfait ces contraintes. Cette approche a été utilisée pour l'apprentissage de la modélisation de bases de données dans Kermit (Suraweera et Mitrovic, 2004) et pour celui de la modélisation orientée objet dans Collect-UML (Baghaei et al., 2006). Dans Collect-UML, l'apprenant doit construire un diagramme de classes à partir d'une description textuelle du problème. Pour modéliser un objet, l'apprenant sélectionne une expression dans l'énoncé. Celle-ci s'affiche alors dans un style différent (gras ou italique) en fonction de l'élément de modélisation choisi. Collect-UML comporte un outil de diagnostic du diagramme de l'apprenant, fondé sur des contraintes syntaxiques et des contraintes sémantiques, propres au domaine considéré. Chaque contrainte est composée d'une condition de pertinence, d'une condition de satisfaction, et d'un message de rétroaction. Lorsque la condition de pertinence est satisfaite, la condition de satisfaction doit l'être aussi : dans le cas contraire, la contrainte est violée, ce qui déclenche le message d'explication associé à cette contrainte. Généralement, l'approche par contraintes est bien adaptée à des systèmes ouverts. Cependant l'efficacité du diagnostic dépend fortement de la définition des contraintes. Or, il est difficile d'identifier des contraintes sémantiques générales s'appliquant aux diagrammes de classe UML. C'est pourquoi, dans Collect-UML, les contraintes sémantiques n'expriment pas les règles du domaine mais les écarts tolérés entre le diagramme de l'apprenant et un diagramme solution fourni par un enseignant. Par exemple, la contrainte n^{o} 49 stipule : « si la solution idéale contient une sous-classe C et que la solution de l'apprenant contient une classe du même nom, alors il est nécessaire que C soit également une sous-classe dans la solution de l'apprenant » et le message d'erreur associé indique : « vérifie que tu as défini toutes le sous-classes requises : il manque certaines sous-classes ». Ceci a de fortes conséquences sur l'interaction, car pour que le système puisse appliquer les contraintes, l'apprenant est obligé de sélectionner les noms des classes et des relations parmi les mots de l'énoncé, sans pouvoir créer d'éléments librement, ni choisir leur nom. De ce fait, Collect-UML est restreint aux énoncés qui ne contiennent pas d'implicites. L'approche par contraintes telle qu'elle est employée ici se rapproche plutôt des « catalogues de bugs », car elle ne permet de reconnaître que les erreurs exhaustivement listées par le concepteur. Comme les contraintes sémantiques servent à vérifier que le diagramme de l'apprenant ne s'écarte pas trop du diagramme solution, les messages de conseil qui leur sont associés tendent à orienter l'apprenant vers la solution du système.

Le système tutoriel DesignFirst-ITS (Moritz, 2008), qui porte sur l'apprentissage de l'analyse et la conception orientée objet, relève d'une autre approche. Il s'appuie sur un modèle du curriculum contenant les concepts de la programmation orientée objet, organisés en un réseau dans lequel les arcs indiquent la nature des relations entre concepts (composant, pré-requis). Un module auteur permet à l'enseignant d'entrer la description d'un problème et de faire générer la solution qui sera utilisée par un module évaluateur. Celui-ci analyse automatiquement la proposition de l'apprenant en la comparant à la solution et à des bugs typiques. Enfin, un agent pédagogique dispense des conseils et des explications. Si DesignFirst-ITS apparaît assez élaboré, il s'apparente plutôt à un système auteur, en raison de la rigidité de l'interaction et des contraintes imposées lors de la création d'un nouvel exercice par l'enseignant. Par exemple, l'énoncé ne doit pas contenir d'implicites ni de détails superflus, et le diagramme solution ne doit pas comporter plus de cinq classes, ce qui limite le champ d'utilisation du système aux toutes premières phases de l'apprentissage.

3. Cadre général de Diagram

Notre problématique porte sur la conception de l'interaction dans un environnement d'apprentissage de la modélisation orientée objet, et plus particulièrement sur les modalités d'action à offrir à l'apprenant et aux rétroactions que doit fournir le système. Nous avons choisi de restreindre le domaine d'étude aux diagrammes de classe UML. Au niveau théorique, nous avons étudié les activités de modélisation dans un cadre d'apprentissage pour élaborer un modèle général d'interaction adapté aux caractéristiques de ce domaine. Afin de pouvoir éprouver nos propositions théoriques, nous avons conçu et implémenté un environnement d'apprentissage appelé Diagram. Enfin, nous avons évalué nos productions en menant

plusieurs expérimentations en contexte écologique.

Avant de présenter de manière détaillée l'environnement Diagram, nous exposons dans ce paragraphe l'approche générale que nous avons retenue, la base d'exercices utilisée pour évaluer l'EIAH, et présentons un exemple de problème qui sera utilisé comme « fil rouge » tout au long de l'article.

3.1. Objectifs et approche retenue

Les exercices de modélisation utilisés dans les premières phases d'apprentissage sont variés, et cette variété est nécessaire pour permettre aux enseignants de construire leur cours en jouant sur la formulation de l'énoncé, les notions UML mises en jeu ou l'introduction progressive des difficultés. Nous avons donc choisi de concevoir Diagram comme un environnement ouvert, dans lequel l'enseignant peut ajouter de nouveaux exercices sans contraintes de vocabulaire ni de nombre d'éléments du diagramme de classes. Les énoncés peuvent contenir des implicites. La création d'un exercice consiste, pour l'enseignant, à saisir un énoncé sous forme textuelle et un diagramme solution à l'aide d'une interface très proche de celle de l'apprenant.

Les études présentées dans la section 2 montrent que deux points sont cruciaux dans la conception de l'interaction avec l'apprenant : d'une part le guidage méthodologique, afin de pallier l'absence de méthode de modélisation, d'autre part les outils de création des éléments du diagramme. Nous avons choisi de mettre en place un guidage méthodologique assez simple, sous forme d'étapes à enchaîner pour créer un diagramme, tout en laissant à l'apprenant la liberté de revenir en arrière en cours de route. Nous avons opté pour l'affichage à l'écran de l'énoncé du problème, comme dans la plupart des autres systèmes, mais nous avons exploité plus largement les possibilités des interfaces graphiques pour renforcer le lien entre texte et diagramme. Enfin, nous avons cherché à soutenir l'auto-correction chez l'apprenant en lui fournissant des aides contextuelles (cf. section 4). De manière générale, notre démarche de conception a été sous-tendue par l'objectif de favoriser les activités cognitives et métacognitives de planification, de contrôle et de vérification, qui jouent un rôle important dans les tâches de modélisation.

A côté de ces aides génériques, les systèmes plus sophistiqués proposent aussi des messages d'aide spécifiques en comparant le diagramme de l'apprenant avec un diagramme solution. DesignFirst-ITS présente l'intérêt de construire automatiquement cette solution à partir de l'énoncé, mais les contraintes qui en résultent sur l'énoncé (texte court, phrases très simples, pas d'implicites) nous ont paru trop fortes et incompatibles avec l'objectif de concevoir un système ouvert. Collect-UML lève partiellement ces restrictions en se basant sur une solution idéale fournie par l'enseignant, mais son approche basée sur les contraintes impose toutefois de limiter les possibilités d'expression à l'interface. Nous avons cherché à dépasser ces limites en explorant une nouvelle voie, toujours basée sur la comparaison de diagrammes - en l'état actuel de la recherche, aucun système n'est capable de se passer d'une solution idéale pour effectuer le diagnostic - mais sans définir *a priori* une liste d'erreurs ou de contraintes. En effet, nous considérons que les différences relevées entre le diagramme de l'apprenant et celui de l'expert ne traduisent pas nécessairement des erreurs, mais peuvent exprimer un autre choix de modélisation. L'analyse des différences entre diagrammes peut se faire indépendamment de l'interprétation de ces différences dans un but pédagogique. Nous avons donc choisi une approche modulaire, en développant séparément un outil qui compare les diagrammes et énumère leurs différences (cf. section 5) et un outil qui interprète ces différences pour fournir des rétroactions pédagogiques à l'apprenant (cf. section 6), en faisant l'hypothèse que cette architecture doit permettre un diagnostic plus robuste et plus pertinent que les autres approches.

3.2. Base d'exercices

Dans une première étape du projet, nous avons constitué une base d'exercices sur les diagrammes de classes afin de les analyser pour mieux caractériser les difficultés rencontrées par les apprenants. Cette base, comprenant une trentaine d'exercices, nous a ensuite servi à mettre au point Diagram, puis à réaliser les expérimentations en contexte écologique. La majorité des exercices a été fournie par Thierry Lemeunier, un enseignant de l'université du Maine qui a participé au projet. Il utilise ces exercices dans ses TD et TP de modélisation orientée objet, avec des étudiants en deuxième année d'université (filière DEUST informatique). Nous avons également utilisé quelques exercices trouvés dans des ouvrages ou sur le web. Les énoncés de cette base concernent des domaines variés, les concepts et le vocabulaire employé sont très divers. Leur taille varie de quelques phrases à une page. Le nombre de classes du diagramme

solution varie de quatre à vingt-sept.

Nous avons analysé cette base pour essayer de produire un classement des exercices en fonction de la difficulté de leur énoncé. Plusieurs critères ont été retenus : la longueur de l'énoncé, la simplicité du vocabulaire employé, la présence d'implicites ou d'informations superflues, la présence d'indices, l'aspect familier du domaine évoqué, le nombre d'informations à modéliser. En combinant ces critères, nous avons obtenu une échelle de difficulté allant de un (pour « très facile ») à cinq (pour « très difficile »). Une analyse similaire a été conduite pour évaluer le degré de difficulté des diagrammes solutions, en se basant sur le nombre de notions UML mises en jeu et leur difficulté. Ces deux classifications ont ensuite été utilisées par planifier les expérimentations en proposant des exercices de complexité croissante.

3.3. Exemple d'énoncé

Nous allons illustrer le fonctionnement de Diagram tout au long de cet article à l'aide d'un exercice intitulé « Stylo et feutre », issu de notre base d'énoncés. La figure 1 présente un diagramme modélisant cet énoncé, que nous utiliserons par la suite comme diagramme de l'expert.

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

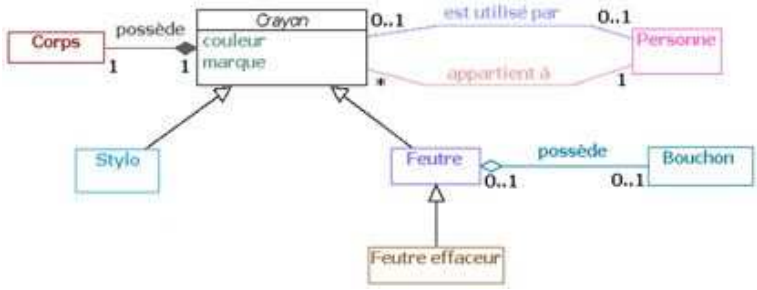


Figure 1 • Diagramme de classes pour l'exercice "Stylo et feutre"

Selon notre classification, cet exercice est considéré comme assez facile (niveau 2) du point de vue de l'énoncé mais assez difficile (niveau 4) du point de vue du diagramme car il met en jeu beaucoup de notions UML. L'énoncé est court, utilise un vocabulaire simple et repose sur des notions familières. Cependant, sa modélisation présente quelques obstacles pour des novices. Une première difficulté provient de ce qu'une des classes est implicite : le terme « crayon » ne figure pas dans l'énoncé. Cet implicite peut dérouter les étudiants qui n'ont traité jusque là que des problèmes dans lesquels toutes les classes sont nommées dans l'énoncé. Nous avons constaté que certains étudiants ne représentent pas cette classe, mais dupliquent tous les attributs et relations qui s'y rapportent. Une deuxième difficulté tient à la présence de deux relations distinctes entre les classes « crayon » et « personne », situation peu courante dans les énoncés simples. Par ailleurs, cet exercice fait appel à quatre types de relations différentes : l'association, la composition, l'agrégation et l'héritage. Or les novices maîtrisent mal les différences entre ces relations et ont tendance à les confondre. Enfin, des erreurs classiques peuvent se manifester sur cet exercice, comme la confusion entre classe et attribut, la mauvaise orientation de relations, ou bien l'indication de multiplicités incorrectes.

L'énoncé fournit certains indices pour guider la modélisation, mais la compréhension de ces indices nécessite elle-même une certaine expérience, et les débutants ne les repèrent pas toujours, ou ne les interprètent pas correctement. Par exemple, l'expression « un corps ayant certaines propriétés » a été introduite par le rédacteur pour indiquer que le corps doit être modélisé comme une classe, et non comme un attribut.

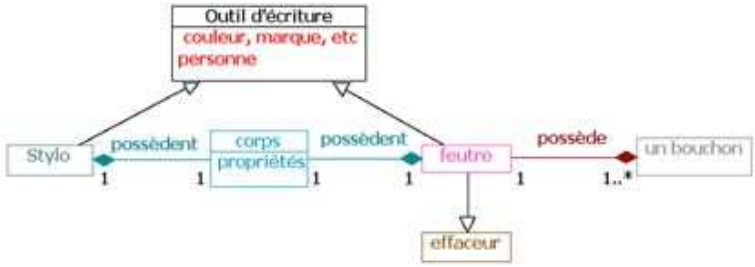


Figure 2 • Exemple de diagramme construit par un étudiant

La figure 2 présente un diagramme créé par un étudiant pour ce même énoncé et contenant des erreurs typiques. Nous l'utiliserons tout au long de l'article comme exemple de diagramme de l'apprenant. Dans cet exemple, l'apprenant a bien repéré la présence d'une classe implicite dans l'énoncé, qu'il a nommée « outil d'écriture ». En revanche, il a dupliqué la composition « possède » pour l'appliquer à la classe stylo et à la classe feutre, plutôt que de la factoriser pour l'appliquer uniquement à la classe « outil d'écriture ». Il a représenté la notion de « personne » comme un attribut de cette classe, au lieu d'en faire une classe à part entière. Les relations entre une personne et un crayon ne sont donc pas explicitées. Le sens de certaines relations est inversé : l'héritage entre « feutre » et « effaceur », la composition entre « feutre » et « bouchon ». Enfin, le diagramme comporte quelques maladresses comme l'intitulé « couleur, marque, etc. » pour un attribut ou l'insertion d'un attribut « propriétés » dans la classe corps.

4. L'interaction dans Diagram

L'introduction d'une forme de guidage méthodologique dans l'interaction peut contribuer à pallier l'absence de méthode de construction d'un diagramme de classes. Nous nous sommes inspirés de la démarche utilisée par l'enseignant qui a participé à la conception de Diagram. Celui-ci recommande à ses étudiants de suivre une méthode en trois étapes : la première consiste à lire entièrement l'énoncé pour se l'approprier, la deuxième consiste à élaborer le diagramme et la troisième consiste à relire une dernière fois l'énoncé en vérifiant l'exactitude du diagramme. Cette méthode est assez simple et générale, elle rejoint les conseils méthodologiques que l'on peut trouver dans des ouvrages sur la modélisation. Nous l'avons transposée dans Diagram en l'enrichissant avec des outils graphiques, soit inspirés des outils papier-crayon, soit permis par l'environnement informatique. Nous l'avons complétée par une quatrième étape (qui sera développée en section 6) dans laquelle le système évalue le diagramme et produit des rétroactions pour aider l'étudiant à le corriger. Le déroulement complet de l'interaction est schématisé en figure 3.

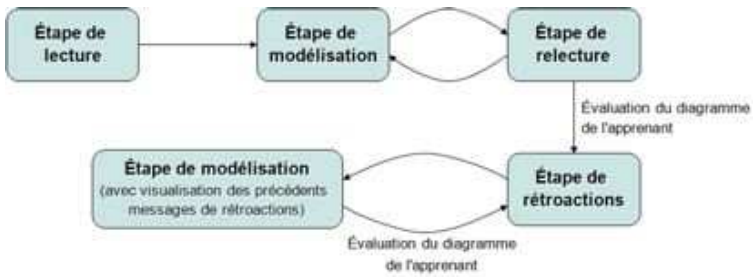


Figure 3 • Étapes de l'interaction dans Diagram

4.1. Étapes de l'interaction

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

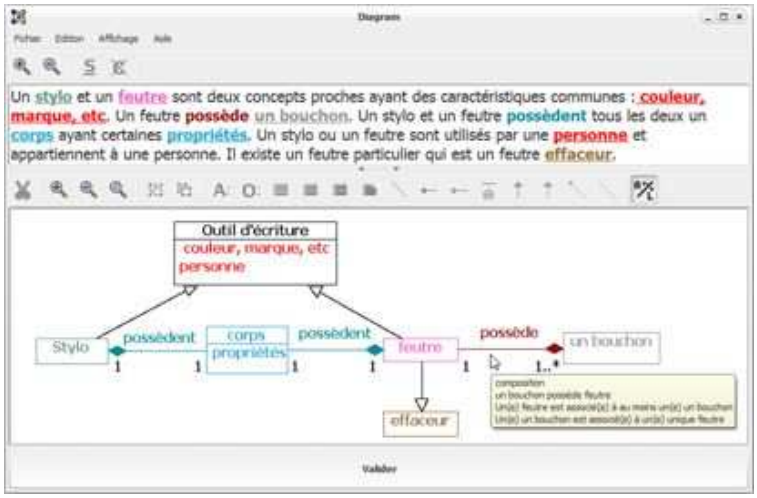


Figure 4 • Interface de Diagram durant l'étape de modélisation

La seconde étape consiste à créer le diagramme de classes correspondant à l'énoncé donné (figure 4). Diagram permet de créer un élément UML (classe, attribut, opération ou relation) de deux manières : directement à partir d'une expression de l'énoncé (« mode assisté » de création) et librement (« mode libre » de création). En mode assisté, l'élément graphique UML modélisé et l'expression sélectionnée dans l'énoncé sont affichés dans une couleur identique, afin de faciliter le contrôle visuel. Cette modalité de création est une originalité de Diagram qui vise à renforcer le lien entre énoncé et modèle. En mode libre, l'apprenant peut créer directement un élément du diagramme de classes en cliquant sur le bouton puis sur le type de l'élément qu'il désire ajouter au diagramme. Cet élément apparaît en noir et l'étudiant lui donne le nom de son choix. Cette modalité permet de créer des éléments qui ne sont pas décrits explicitement dans l'énoncé ainsi que des éléments n'ayant pas de nom, comme les relations d'héritage. L'étudiant a ensuite la possibilité de lier entre elles les expressions représentant un même concept dans l'énoncé en sélectionnant une expression et en la rattachant directement à un élément graphique existant ou à une autre expression à partir du menu contextuel. Les expressions rattachées prendront la même couleur que l'élément sélectionné. L'étudiant est libre de modifier les noms des éléments créés tout au long de l'activité de modélisation.

Durant la troisième étape, l'apprenant doit relire l'énoncé de l'exercice et le comparer à son diagramme de classes afin de vérifier que le modèle est complet et correct par rapport à l'énoncé. Au début de cette phase, l'interface ne présente plus que l'énoncé, en noir et blanc. Au fur et à mesure que l'étudiant passe la souris sur le texte, les expressions modélisées deviennent à nouveau colorées ; simultanément, les éléments correspondants dans le diagramme réapparaissent à l'interface. L'étudiant peut alors revenir à la phase de modélisation, s'il veut modifier son diagramme, ou bien passer à l'étape d'évaluation par le système, s'il estime que son diagramme est terminé.

4.2. Aides contextuelles

Cette étape intègre différentes aides contextuelles qui favorisent le contrôle de l'activité. En effet, nous avons constaté que la signification des éléments du diagramme n'est pas toujours bien appréhendée par les novices. Faisant l'hypothèse que confronter l'apprenant à une paraphrase de son diagramme peut l'aider à mieux percevoir sa signification, nous avons intégré à l'interface un dispositif de reformulation textuelle des éléments du diagramme. Lorsque la souris pointe sur un élément du diagramme (classe ou relation), un message affiché dans une infobulle indique la signification de cet élément sous forme textuelle. La figure 4 présente la reformulation affichée pour la relation « possède » de notre exemple. Ce dispositif a été expérimenté et nous avons montré qu'il favorisait l'auto-correction chez l'utilisateur (Alonso et al., 2008).

5. Analyse du diagramme de l'apprenant

Les aides prévues dans le cadre général d'interaction décrit ci-dessus sont génériques, au sens où elles ne tiennent pas compte de la validité du diagramme. Pour produire des rétroactions plus spécifiques, basées

sur l'analyse des productions individuelles de l'étudiant, il est nécessaire de comparer le diagramme à une solution de référence, comme le font Collect-UML ou DesignFirst-ITS (Baghaei et al., 2006) (Moritz, 2008). Cependant, les méthodes de diagnostic employées dans ces deux systèmes imposent des restrictions fortes sur les énoncés et les diagrammes. Nous avons exploré une autre approche, plus compatible avec le caractère ouvert de Diagram. Elle consiste à repérer toutes les différences existant au niveau domaine entre les deux diagrammes, à l'aide d'un outil générique de comparaison de diagrammes, puis à exploiter ces différences pour produire des rétroactions au niveau pédagogique. Nous présentons dans cette section la méthode de comparaison de diagrammes en donnant un exemple d'application.

5.1. Méthode de comparaison des diagrammes

À un niveau général, la comparaison de deux diagrammes s'apparente à un processus d'appariement de modèles. L'appariement consiste à identifier et qualifier des relations entre les éléments de plusieurs modèles. Le résultat du processus est un alignement des éléments des modèles, c'est-à-dire un ensemble de correspondances entre deux ou plusieurs éléments (Shvaiko et Euzenat, 2005).

De nombreuses techniques ont été synthétisées et expérimentées dans les domaines de l'appariement de schémas (Rahm et Bernstein, 2001), d'ontologies (Euzenat et Shvaiko, 2007) et de graphes (Sorlin et al., 2007). Cependant, elles ne sont pas directement applicables à la comparaison de diagrammes UML. En nous inspirant de ces techniques, nous avons défini une méthode automatique d'appariement, appelée ACDC (*Automatic Class Diagrams Comparator*), qui prend en entrée des diagrammes de classes UML et produit en sortie un alignement de leurs constituants. Les critères utilisés pour comparer et mettre en correspondance les diagrammes sont extraits directement de la sémantique du métamodèle des diagrammes de classes UML.

La méthode ACDC suit trois étapes que nous allons détailler successivement :

1. Schématisation des diagrammes en motifs,
2. Évaluation des similarités et des différences,
3. Choix de l'appariement des motifs et des différences.

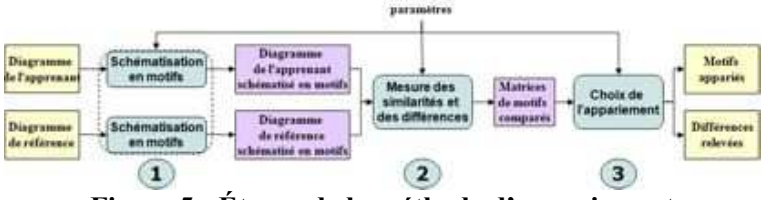


Figure 5 • Étapes de la méthode d'appariement

5.2. Schématisation des diagrammes en motifs

Afin d'exploiter les particularités des modèles à apparier, nous avons introduit dans la méthode de comparaison des structures caractéristiques appelées *motifs* qui correspondent aux différents niveaux de granularité d'un modèle. Les motifs simples sont les éléments du diagramme. Ces motifs sont organisés et structurés par des motifs complexes, c'est-à-dire des ensembles d'éléments formant des constructions possédant une sémantique et une structure bien définies. Dans le cas des diagrammes de classes UML, les motifs sont extraits du métamodèle UML (figure 6). Les motifs simples sont les classes, les associations, les héritages, les attributs, etc. Les motifs complexes sont les chaînes d'associations et les hiérarchies d'héritage qui sont construites à partir de classificateurs et d'un type de relation, respectivement l'association et l'héritage.

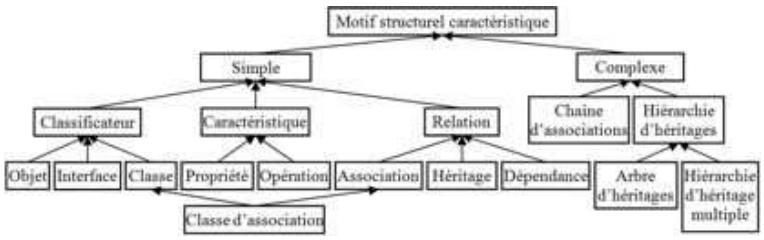


Figure 6 • Motifs structurels caractéristiques des diagrammes de classes UML 2.x

L'intérêt de ces motifs complexes est que leurs propriétés structurelles et sémantiques peuvent être exploitées pour apparier les diagrammes à un niveau de granularité élevé, avant d'apparier les éléments les plus simples. Cela permet d'orienter la comparaison et de trouver plus vite une solution. En effet, la comparaison de quelques motifs complexes est plus rapide que celle des nombreux motifs simples qui les composent. De plus, le processus de propagation dans les hiérarchies d'héritages permet de repérer des déplacements d'éléments, tels que le déplacement d'un attribut d'une classe à une autre, ou le déplacement d'une relation d'une classe mère vers sa classe fille, ce qui serait très difficile à faire sans analyser le diagramme à un niveau abstrait en s'appuyant sur les motifs.

La schématisation en motifs des diagrammes de notre exemple est illustrée par les figures 7 et 8. Sur le diagramme de référence, les motifs simples sont structurés en trois motifs complexes : deux chaînes d'associations (les rectangles) et un arbre d'héritage (le triangle). Le diagramme de l'apprenant comporte une seule chaîne d'associations et deux arbres d'héritage. Ces deux arbres d'héritage forment une hiérarchie d'héritage multiple dont les racines sont les classes "Outils d'écriture" et "effaceur".

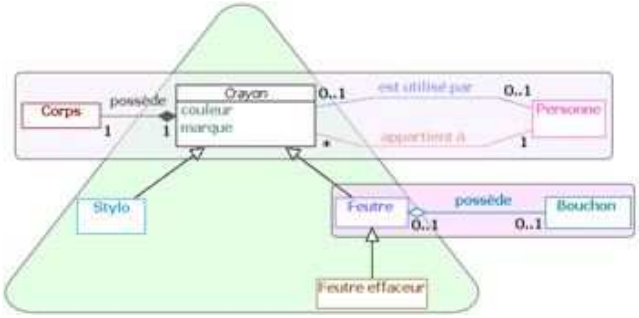


Figure 7 • Diagramme de référence schématisé en motifs

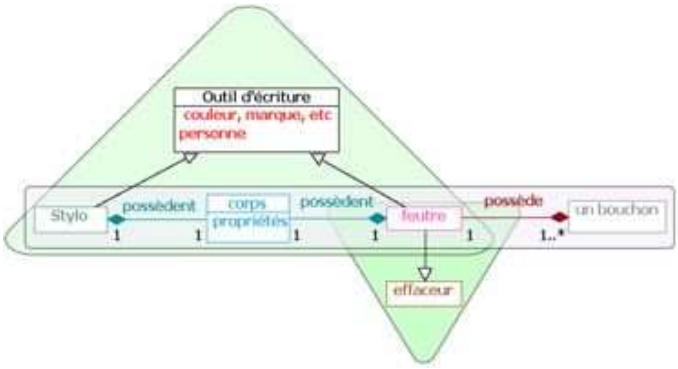


Figure 8 • Diagramme de l'apprenant schématisé en motifs

5.3. Évaluation des similarités et des différences

Une fois que les diagrammes ont été schématisés en motifs, leurs similarités et différences sont évaluées. Le premier objectif de cette phase est d'affecter un score de similarité à chaque couple de motifs comparés par type pour pouvoir classer les motifs par ordre de ressemblance. Le second objectif est de relever les différences de structuration entre les motifs « en contexte » de chaque couple de motifs comparés (nous appelons motifs « en contexte » les motifs contenus, conteneurs et liés au couple comparé).

Pour cela, nous avons défini une mesure évaluant les similarités et les différences des motifs par couple d'un même type. Toutes les dimensions descriptives des motifs participent *a priori* à l'expression de la similarité. La mesure repose sur le principe suivant : deux motifs sont d'autant plus similaires que les motifs directement reliés à ceux-ci le sont aussi, selon un principe de renforcement mutuel. Mais une telle définition mène à une dépendance mutuelle et récursive dans les calculs. Par exemple, les scores de similarité de deux classes et de leurs attributs dépendent les uns des autres. Pour pallier ce problème, nous avons défini une fonction de calcul de la similarité d'un couple de motifs combinant deux scores disjoints. Le *score simple* pondère des critères de similarité indépendants de tout autre motif des

diagrammes. Le nom, l'abstraction, le nombre de relations liées à un élément sont des critères intervenant dans le calcul du score simple. Le *score complexe* agrège une partie des scores des motifs en contexte avec le couple de motifs comparés.

La dénomination des éléments dans un diagramme (classes, attributs, opérations et relations) est un critère de notre mesure. Mais comme les éléments se réfèrent à des objets du monde ou des concepts, l'étudiant peut utiliser de nombreux synonymes à la place des noms employés dans le diagramme de référence. De plus, les étudiants font des fautes de frappe et d'orthographe. Un appareur spécifique aux chaînes de caractères a donc été mis en place. Il se charge de rechercher les sous-chaînes communes des espaces de nommage (c'est-à-dire ici l'ensemble des noms donnés aux éléments du diagramme). Au préalable, les espaces de nommage sont filtrés par casse, genre et nombre. De plus, les caractères spéciaux et les séparateurs y sont supprimés.

L'ensemble de ce dispositif permet de repérer des correspondances entre éléments non identiques des deux diagrammes. Ainsi, dans notre exemple, l'apprenant a représenté un seul attribut « couleur, marque, etc. » là où le diagramme de référence en comporte deux : « marque » et « couleur ». Le traitement effectué par l'appareur permet de repérer la proximité entre les noms de ces attributs et de mettre en correspondance les deux attributs du diagramme de l'apprenant avec l'attribut unique du diagramme de référence. D'autre part, l'apprenant a nommé « Outil d'écriture » la classe appelée « Crayon » dans le diagramme de référence. Ces deux noms sont trop éloignés pour que l'appareur de chaînes puisse détecter une similarité : dans ce cas c'est la prise en compte du contexte dans le calcul du score complexe qui permet de mettre en correspondance les deux classes, car elles ont des contextes très similaires, notamment des classes filles en commun.

La mesure de similarité est instanciée sous la forme d'une hiérarchie de comparateurs (Auxepaules, 2009) prenant en charge l'évaluation des similarités et des différences locales aux modèles, aux motifs et aux chaînes de caractères.

5.4. Choix des appariements de motifs

Une fois les scores de similarité calculés et attribués à chaque couple de motifs, il reste à choisir un alignement, c'est-à-dire un ensemble de correspondances entre les éléments des deux diagrammes. Chaque correspondance peut être univoque ou multivoque. Dans le contexte des graphes, un appariement est dit univoque lorsque chaque sommet d'un graphe est associé avec au plus un sommet de l'autre graphe. Un appariement est dit multivoque lorsque chaque sommet d'un graphe peut être associé à un ensemble de sommets de l'autre graphe (Sorlin et al., 2007). Dans le cas des diagrammes, la correspondance sera dite univoque quand elle met en relation un élément de chacun des diagrammes, et multivoque quand elle met en relation un élément d'un diagramme avec plusieurs éléments de l'autre diagramme. Pour illustrer ceci sur notre exemple, on repère un appariement univoque entre les classes « feutre » de chaque diagramme, et un appariement multivoque entre d'une part l'attribut « couleur, marque, etc. » du diagramme de l'apprenant et les deux attributs « marque » et « couleur » du diagramme de référence.

Dans notre proposition, le processus d'appariement est mené de manière gloutonne, des structures les plus générales vers les structures les plus spécifiques (*top-down*). Un algorithme *top-down* est en général moins coûteux qu'un algorithme *bottom-up*, car l'appariement à un niveau élevé de la structure du modèle restreint les choix d'appariement des structures plus fines (Madhavan et al., 2001). Dans le cas de nos diagrammes, l'alignement des motifs complexes est construit en premier, puis celui des classes, ensuite celui des relations et enfin celui des caractéristiques. Nous avons retenu un processus glouton en raison des contraintes de temps : le système doit être suffisamment rapide pour produire des rétroactions pédagogiques synchrones, le délai de réponse ne doit pas dépasser quelques secondes. Ce choix présente des inconvénients : un algorithme glouton n'autorise pas de retour en arrière lorsqu'un choix s'avère non optimal et il ne permet pas de mémoriser des solutions alternatives. Cependant, la qualité des alignements produits par notre méthode, sans être optimale, reste satisfaisante dans le contexte de Diagram (une évaluation détaillée de la qualité des résultats produits par ACDC est disponible dans (Auxepaules, 2009) et (Auxepaules, 2010)).

5.5. Recensement des différences structurelles

Le résultat produit à l'issue du processus est une liste de couples de motifs appariés, où chaque couple peut être étiqueté par une ou plusieurs différences. Lorsque les deux motifs du couple sont identiques, aucune différence n'est notée. Dans le cas contraire, les motifs peuvent différer soit par leur structure générale ou leur agencement dans le diagramme, nous parlons alors de *différence générale*, soit par des caractéristiques internes, nous parlons alors de *différence spécifique*.

Les différences spécifiques sont liées aux propriétés et à la sémantique des modèles et varient en fonction du métamodèle des modèles comparés. Pour les diagrammes de classes UML, les différences spécifiques des motifs simples portent notamment sur le nom, l'abstraction, l'orientation des relations et celles des motifs complexes sont liées à la propagation des propriétés et des relations d'une classe mère vers ses filles ou inversement.

Les différences sont liées aux appariements. Elles sont donc aussi univoques ou multivoques. Les différences univoques élémentaires sont l'omission et l'insertion d'un motif. Toutes les autres différences sont construites par combinaison de ces deux différences de base. On obtient ainsi deux différences univoques plus complexes, le transfert (déplacement d'un élément d'un endroit à un autre) et le remplacement (substitution d'un élément par un autre, au même endroit). En combinant les différences univoques, on obtient trois différences multivoques : un motif du diagramme d'origine peut être apparié avec plusieurs motifs de l'autre diagramme (éclatement), ou inversement (fusion), et plusieurs groupes de motifs des deux diagrammes peuvent être appariés (regroupement).

La taxinomie des différences produite par ACDC, que nous appelons taxinomie des différences structurelles (TDS), est présentée dans la figure 9.

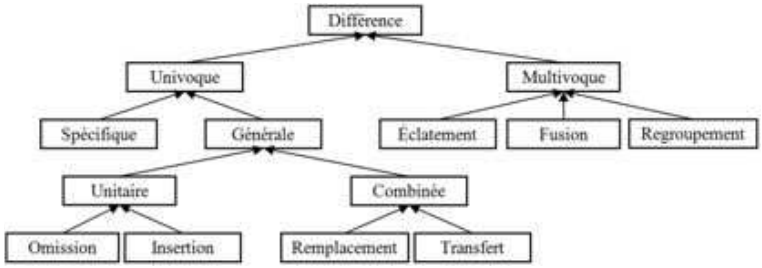


Figure 9 • Taxinomie des différences structurelles

5.6. Exemple

Nous présentons dans cette section les résultats de l'appariement pour l'exemple de l'exercice "Stylo et feutre". Nous détaillons les principales différences et donnons pour les trois premières les sorties correspondantes de notre système. Chaque ligne est de la forme : élément du diagramme de l'apprenant (entre accolades), mot-clé désignant le type de différence, élément du diagramme de référence (entre accolades). L'apprenant a représenté la notion de personne sous forme d'un attribut de la classe « Outil d'écriture », au lieu d'en faire une classe. Ceci se traduit par une différence de remplacement d'une classe par un attribut. Une différence spécifique précise que les éléments sont de nature différente. L'absence des relations « appartient à » et « est utilisé par » dans le diagramme de l'apprenant est indiquée par deux différences univoques d'omission.

{Outil d'écriture:personne} REMPLACEMENT {Personne}

{Outil d'écriture:personne} NATURE_INCOMPATIBLE {Personne}

OMISSION {appartient à (Personne-Crayon)}

OMISSION {est utilisé par (Personne-Crayon)}

L'apprenant a représenté deux relations de composition « possèdent » s'appliquant aux classes filles de « Outil d'écriture », alors que le diagramme de référence ne comporte qu'une composition, appliquée à la classe mère. Ceci se traduit par une différence d'éclatement de la composition, ainsi que deux différences de transfert des deux compositions vers les classes filles.

{possèdent(corps-feutre) possèdent (corps-Stylo)} ECLATEMENT

{possède (Corps-Crayon)}

{possèdent (corps-feutre)} TRANSFERT_A_FILLE possède (Corps-Crayon)}

{possèdent (corps-Stylo)} TRANSFERT_A_FILLE possède (Corps-Crayon)}

L'apprenant a remplacé l'agrégation « possède » entre « un bouchon » et « feutre » par une composition. De plus, l'orientation de la relation est inversée. Les deux relations sont appariées et deux différences spécifiques (sur la même ligne) indiquent respectivement que les relations mises en correspondance sont différentes et que l'orientation est inversée. Les deux lignes suivantes indiquent les différences spécifiques relatives aux multiplicités de ces relations. Par ailleurs, le sens de l'héritage entre les classes « effaceur » et « feutre » a été inversé : les deux relations d'héritage sont appariées et une différence spécifique indique l'inversion de sens.

{possède (feutre-un bouchon)} APPARIEMENT {possède (Bouchon-Feutre)}

{possède (feutre-un bouchon)} COMPOSITION_EN_AGREGATION INVERSE

{possède (Bouchon-Feutre)}

{un bouchon_possède} MULTIPLICITES_TERMINENT {Bouchon_possède}

{feutre_possède} MULTIPLICITES_RENCONTREES_PAR {Feutre_possède}

{(effaceur<-feutre)} APPARIEMENT {(Feutre<-Feutre effaceur)}

{(effaceur<-feutre)} INVERSE {(Feutre<-Feutre effaceur)}

L'apprenant a représenté la classe abstraite « Crayon » sous la forme d'une classe concrète nommée « Outil d'écriture ». Une différence spécifique décrit la différence d'abstraction entre ces deux classificateurs.

Dans le diagramme de l'apprenant, les attributs « marque » et « couleur » de la classe abstraite « Crayon » sont représentés sous forme d'un attribut nommé « couleur, marque, etc. ». Une différence multivoque de fusion entre ces différents attributs est repérée.

Enfin, l'apprenant a affecté un attribut « propriétés » à la classe « corps ». Le diagramme de référence ne contenant aucun élément correspondant, cette différence est considérée comme une insertion.

6. Rétroactions pédagogiques

La comparaison entre le diagramme de l'apprenant et le diagramme de référence produit une liste exhaustive de différences qui sont basées sur le métamodèle UML et expriment des écarts entre les structures et les caractéristiques des diagrammes. De telles différences ne sont pas exploitables directement au plan pédagogique, car elles n'ont pas été conçues dans ce but. Pour produire des rétroactions qui soient pertinentes du point de vue de l'apprentissage, il est nécessaire de définir une seconde taxinomie de différences, à visée pédagogique, puis d'élaborer un mécanisme de conversion entre les deux représentations. Une fois cette conversion effectuée, un message de rétroaction peut être élaboré pour traiter chaque différence.

6.1. Taxinomie des différences pédagogiques

Nous avons élaboré une seconde liste de différences que nous appelons *différences pédagogiques* pour les distinguer des différences structurelles. Dans un premier temps nous avons recueilli les diagrammes produits par les apprenants lors d'expérimentations réalisées avec Diagram à l'automne 2006 et en mars 2007, avant l'intégration de l'outil de comparaison de diagrammes (Alonso et al., 2008). Nous avons sélectionné deux exercices pour lesquels nous avons obtenu un grand nombre de diagrammes produits par les apprenants (respectivement 26 et 30 diagrammes). Selon notre classification des exercices (cf. 3.2), le premier exercice présente un énoncé assez facile et un diagramme assez difficile, le second présente un énoncé et un diagramme de difficulté moyenne. A eux deux, ces exercices contiennent l'ensemble des notions du domaine couvert par Diagram.

Pour chaque exercice, nous avons comparé la solution de l'apprenant avec le diagramme de référence

produit par un enseignant. Nous avons ainsi obtenu une liste de différences considérée comme exhaustive. À partir de cette liste, nous avons établi une taxinomie de différences « simples » qui sert de base à la construction des rétroactions pédagogiques. En effet, chaque différence simple permet la construction d'un message de rétroaction. Nous avons regroupé les différences simples en huit catégories :

- omission d'un élément : l'apprenant a omis un élément du diagramme de référence ;

- ajout d'un élément : l'apprenant a ajouté dans son diagramme un élément qui ne figure pas dans le diagramme de référence ;

- transfert d'un élément : un élément a été transféré dans une autre partie du diagramme. Par exemple une relation entre deux classes A et B dans le diagramme de référence est déplacée par l'apprenant entre les classes A et C ;

- dédoublement d'un élément : un élément du diagramme de référence est représenté dans le diagramme de l'apprenant par plusieurs éléments du même type ;

- fusion d'éléments : plusieurs éléments d'un même type sont représentés dans le diagramme de l'apprenant par un seul élément ;

- représentation erronée : un élément du diagramme de référence est représenté dans le diagramme de l'apprenant mais sous une autre forme. Par exemple, une classe est remplacée par un attribut, une composition par une association, etc.

- inversion du sens d'une relation : le sens d'une relation orientée (héritage, agrégation ou composition) a été inversé par l'apprenant ;

- multiplicité erronée : les multiplicités d'une relation dans le diagramme de l'apprenant comportent des différences avec celles du diagramme de référence.

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

Lorsque toutes les différences simples ont été identifiées, on s'attache à repérer les combinaisons d'erreurs. Pour chaque différence principale, on recherche la présence d'une ou plusieurs différences associées. Les différences combinées que nous obtenons seront prioritaires sur les différences simples qui les composent, et donneront lieu à une rétroaction spécifique. Les différences simples restantes, qui se sont produites de manière isolée, donneront lieu à une rétroaction séparée.

6.2. Correspondance entre les deux taxinomies

Après avoir établi la taxinomie de différences simples et composées, nous nous sommes intéressés à la correspondance entre les deux taxinomies, TDS et TDP. En effet, le diagnostic produit une liste de différences en terme d'appariement de motifs structurels, et ces différences ne sont pas utilisables directement pour construire les rétroactions. Pour chaque diagramme réalisé par un étudiant, nous avons 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 taxinomies.

Différence structurelle (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

Tableau 1 • Correspondance entre les deux taxinomies

6.3. Organisation des rétroactions

La comparaison effectuée entre le diagramme de l'apprenant et le diagramme de référence produit une liste de différences. Celles-ci ne traduisent pas nécessairement une erreur, elles peuvent signifier simplement que l'étudiant a effectué un autre choix de modélisation que celui du diagramme de référence. Les messages de rétroactions dans Diagram ne sont donc pas des « messages d'erreur » classiques, ils visent à attirer l'attention de l'étudiant sur certaines parties de son diagramme afin qu'il les vérifie, et éventuellement les corrige. Dans certains cas, la différence observée suggère que l'apprenant ne maîtrise pas certaines notions du langage UML : la rétroaction est alors complétée par un rappel de cours.

La méthode de diagnostic par comparaison, développée dans Diagram, nécessite que l'apprenant ait achevé son diagramme. Sur un diagramme inachevé, les éléments non encore modélisés ne peuvent pas être distingués des éléments omis, et le diagnostic les considérerait comme des omissions. Nous avons donc choisi d'intégrer la phase d'analyse du diagramme à l'issue de l'étape de relecture. L'apprenant peut ensuite revenir à la phase de construction pour modifier son diagramme en fonction des rétroactions reçues (cf. figure 3).

6.4. Formulation des rétroactions

Selon les cas, une différence repérée entre les diagrammes peut traduire une erreur ou bien une simple variante de représentation de l'énoncé. Le texte de la rétroaction doit donc être modulé selon le degré de certitude sur la présence d'une erreur.

En nous appuyant sur la classification proposée par (Lemeunier, 2000), nous avons retenu trois formes d'intervention : indiquer, questionner et proposer. « Indiquer » consiste à attirer l'attention de l'étudiant sur une partie du diagramme, ou sur la manière dont il a représenté une notion de l'énoncé. Cela peut consister en une reformulation textuelle d'un élément du diagramme, visant à faire percevoir à l'apprenant que la représentation qu'il a choisie peut poser question. « Questionner » consiste à interroger l'apprenant sur les caractéristiques de son diagramme. Les questions incitent l'apprenant à vérifier mentalement que le diagramme satisfait une propriété donnée. La modalité « proposer », la plus directe, consiste à suggérer la correction à apporter au diagramme : elle ne s'applique que lorsque la présence d'une erreur est quasi-certaine.

Selon les types de différence, une ou plusieurs modalités d'intervention seront appropriées. Ainsi, en cas d'ajout ou d'omission d'éléments, qui ne constituent pas nécessairement des erreurs, on utilise les modalités « indiquer » et « questionner ». Dans certains cas de représentation incorrecte d'une relation (par exemple, un héritage au lieu d'une composition), on peut également utiliser la modalité « proposer »

car la présence d'une erreur est très probable. Au cours de l'interaction, les interventions sont graduées de la plus générale (indiquer) à la plus précise (proposer) : pour chaque différence observée, le message le plus général est affiché, et si l'apprenant estime que cela ne suffit pas, il peut faire apparaître le message de niveau inférieur.

6.5. Exemple de rétroactions

Nous présentons dans cette section les rétroactions produites à partir du diagramme d'un apprenant pour l'exemple « stylo et feutre ». Les différences structurelles identifiées au cours de l'appariement (cf. section 5.6) ont été converties en différences pédagogiques simples. Parmi celles-ci, trois sous-ensembles sont reconnus comme formant une différence composée (A, B et C), les autres sont des différences simples isolées (D, E, F et G).

Différences pédagogiques simples	Différences pédagogiques composées
Représentation erronée de la classe "Personne" par un attribut "personne" de la classe "Crayon"	(A) Représentation erronée d'une classe et omission des éléments liés à cette classe
Omission de la relation "appartient à" entre les classes "Personne" et "Crayon"	
Omission de la relation "est utilisé par" entre les classes "Personne" et "Crayon"	
Dédoublement de la relation "possède" liant les classes "Corps" et "Crayon"	(B) Dédoublement et transfert d'une relation
Transfert de la relation "possède" de "Crayon" vers "Feutre"	
Transfert de la relation "possède" de "Crayon" vers "Stylo"	
Inversion du sens de la relation "possède" entre les classes"Bouchon" et "Feutre"	(C) Représentation erronée d'une relation et inversion de sens
Type erroné de la relation "possède" entre "Bouchon" et "Feutre"	
Multiplicité erronée de "possède" du côté de "Feutre"	
Multiplicité erronée de "possède" du côté de "Bouchon"	
(D) Inversion du sens de la relation d'héritage entre les classes "Feutre" et "Feutre effaceur"	
(E) Représentation de la classe abstraite "Crayon" par une classe ordinaire	

(F) Fusion des attributs "couleur" et "marque" en un seul	
(G) Ajout d'un attribut "propriétés" à la classe "Corps"	

Tableau 2 • Différences pédagogiques simples et composées

Pour chacune des ces sept différences pédagogiques, un message de rétroaction est produit selon les modalités définies dans le paragraphe précédent. Nous en donnons trois exemples

La première composée A indique la représentation d'une classe sous forme d'un attribut et l'omission des éléments liés à cette classe. À partir de cette différence, nous obtenons une rétroaction pédagogique selon trois modalités. La rétroaction se focalise sur la représentation erronée de la classe « Personne », sans mentionner l'absence des relations puisque celles-ci portent sur une classe que l'apprenant n'a pas représentée.

Indiquer : Tu as représenté **personne** comme un attribut de la classe **Outil d'écriture**.

Questionner : L'attribut **personne** peut-il avoir des propriétés et/ou des comportements ?

Proposer : Si **personne** possède des propriétés et/ou des comportements, c'est plutôt une classe.

La différence composée C est formée d'une erreur de type et d'une inversion de sens (les erreurs de multiplicité sont considérées comme secondaires par rapport à ces erreurs, et sont masquées). La rétroaction associée est focalisée sur la différence d'inversion du sens de relation.

Indiquer : Tu dis que **un bouchon** est composé(e) de **feutre**.

Questionner : Est-ce que **un bouchon** est composé(e) de **feutre** ?

Proposer : Je dirais plutôt que c'est **feutre** qui est composé(e) de **un bouchon**.

Enfin, la différence pédagogique simple G indique que l'apprenant a ajouté un attribut "propriétés" à la classe "corps". Cette différence conduit à une rétroaction selon deux modalités. La modalité de proposition est absente ici, car l'insertion d'un attribut n'est pas nécessairement une erreur.

Indiquer : Ta classe **corps** possède l'attribut **propriétés**.

Questionner : Est-ce que **propriétés** est un attribut de **corps** ?

7. Expérimentation

Les premières versions de Diagram ont fait l'objet d'expérimentations ciblées sur les aides contextuelles (Alonso et al., 2008), et le module de diagnostic a été évalué séparément (Auxepaules, 2010). Nous présentons uniquement ici l'expérimentation menée après l'intégration du module de diagnostic, qui se focalise sur la pertinence des diagnostics effectués en situation réelle et sur l'effet des messages de rétroaction sur le comportement des apprenants.

7.1. Méthode d'évaluation

L'expérimentation s'est déroulée à l'automne 2008 avec dix-huit étudiants de deuxième année de DEUST Informatique Systèmes et Réseau qui ont travaillé avec Diagram durant quatre séances de trois heures. L'expérimentation s'est déroulée en conditions écologiques, c'est-à-dire durant des séances de TP intégrées au cursus des étudiants. 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 convaincus 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.

7.2. Analyse des séances

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. Pour chaque exercice, nous avons compté le nombre d'appels au diagnostic, le nombre de messages et le nombre de messages lus. Un message est considéré comme lu si l'apprenant lit 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 au 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 %. Pour remédier à cette difficulté, il serait possible de hiérarchiser les messages et de n'afficher, dans un premier temps, que les messages les plus prioritaires.

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 lus correspondent à 565 messages distincts.

Sur ces 565 messages, on constate dans 11,6% des cas que 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 résultent sont donc incohérents. Ces diagnostics inappropriés sont dus à des appariements incorrects entre les éléments des deux diagrammes. Cela peut se produire localement, sur certaines parties du diagramme qui diffèrent à la fois par leur structure et par le nom des éléments. En effet, le processus d'appariement s'appuie sur ces deux aspects : si aucune similarité de nom ou de structure ne peut être trouvée, l'appariement sera de mauvaise qualité (Auxepaules, 2010). Toutefois, cela n'a pas perturbé les séances car les incohérences étaient assez flagrantes et les étudiants ont généralement choisi d'ignorer ces messages.

Nous avons étudié les réactions des apprenants à la suite des messages cohérents et distingué trois cas.

- Correction : l'apprenant modifie son diagramme de manière cohérente avec le message (49,7 %).

- Modification : l'apprenant effectue une modification en lien avec le message mais son diagramme reste différent du diagramme de référence (3,8 %).

- Sans effet : l'apprenant ne modifie pas son diagramme (46,5 %).

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 des 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 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 3 indique le pourcentage de chaque différence pédagogique parmi les messages lus et le taux de correction par différence.

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	4,61	86,96 %
Fusion	2,81	21,43 %
Inversion de sens	1,20	100,00 %
Transfert	0,20	100,00 %
Total	100,00	

Tableau 3 • Taux de correction des différences

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 indiquent précisément la modification à effectuer. Le taux de correction montre que, dans la moitié des cas, les étudiants réalisent la modification attendue. 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. À 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.

Nous avons observé que, dans certains cas, la lecture d'un message et la correction du diagramme conduisent l'apprenant à vérifier et corriger d'autres éléments, proches de ceux concernés par le message. Cela suggère que les messages de rétroaction favorisent de manière effective l'auto-vérification chez l'apprenant.

Globalement, le temps passé sur Diagram après le premier appel à l'outil de 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.

8. Conclusion

Nous avons présenté une nouvelle approche pour la conception d'un EIAH dédié aux diagrammes de classe. Cette approche contribue à surmonter certaines limites des systèmes existants : elle combine un environnement ouvert, qui n'impose pas de contraintes sur l'énoncé, comme dans les éditeurs UML à but pédagogique, et la production de rétroactions spécifiques au diagramme de l'apprenant, comme dans les EIAH plus sophistiqués. La démarche modulaire adoptée, avec l'intégration d'un outil d'appariement de diagrammes, permet de remédier à l'incomplétude des catalogues de bugs ou de contraintes, puisque l'appariement produit la liste exhaustive des différences entre les diagrammes. Elle facilite la production de rétroactions selon différents niveaux de granularité grâce à la notion de différence simple et composée.

Plusieurs expérimentations en contexte écologique ont montré que le modèle d'interaction embarqué dans Diagram encourage les activités de régulation chez les apprenants, en favorisant la vérification et l'auto-correction.

Nous nous sommes focalisés sur l'apprentissage en phase d'initiation, et cela impose des limites à ce travail. Pour des étudiants plus avancés, les caractéristiques de l'interface et de l'interaction (découpage en étapes, usage des couleurs) ne seraient plus pertinentes, et il serait nécessaire d'introduire d'autres types de diagrammes UML pour élargir le domaine d'application. En revanche, il serait intéressant d'éprouver la généricité de ce cadre d'interaction en l'appliquant à un domaine voisin, comme la modélisation de schémas entités-relations dans les bases de données : le découpage en étape, les outils de modélisation et les aides pourraient être adaptés.

L'appariement des diagrammes repose sur un algorithme glouton, qui n'autorise pas le retour-arrière et peut conduire à des résultats de faible qualité. Pour y remédier, il faudrait mettre en place un algorithme de recherche locale taboue réactive, ou de recherche heuristique. Par ailleurs, le système ne dispose que d'une solution de référence et n'est pas en mesure de reconnaître une solution alternative correcte proposée par l'apprenant. Pour surmonter cette limite, il serait nécessaire d'adapter l'algorithme de diagnostic afin qu'il prenne en compte des variantes de la solution de référence. Mais cela requiert de décrire les relations entre des diagrammes ayant des parties en commun et de fusionner les résultats des appariements, ce qui est considéré actuellement comme un problème très complexe et mal résolu.

Enfin, une limite des aides provient de ce que les rétroactions sont produites de manière indépendante à chaque appel du diagnostic. On pourrait chercher à mémoriser les diagnostics successifs, afin d'éviter de répéter plusieurs fois le même message, ou à l'inverse d'insister si l'apprenant répète une erreur de manière récurrente.

Bibliographie

ALONSO M., PY D., LEMEUNIER T. (2008). A Learning Environment for Object-Oriented Modelling, Supporting Metacognitive Regulations. *Proceedings of the Eighth IEEE International Conference on Advanced Learning Technologies* (ICALT'08). Santander, Spain.

ALONSO M.. (2009). *Conception de l'interaction dans un EIAH pour la modélisation orientée objet*. Thèse de doctorat en informatique, Université du Maine, Le Mans, France, 164 p. Disponible sur Internet : http://cyberdoc.univ-lemans.fr/theses /2009/2009LEMA1007.pdf

AUXEPAULES L., PY D., LEMEUNIER T. (2008). A Diagnosis Method that Matches Class Diagrams in a Learning Environment for Object-Oriented Modelling. *Proceedings of the Eighth IEEE International Conference on Advanced Learning Technologies* (ICALT'08). Santander, Spain.

AUXEPAULES L. (2009). *Analyse des diagrammes de l'apprenant dans un EIAH de la modélisation orientée objet*. Thèse de doctorat en informatique, Université du Maine, Le Mans, France, 226 p. Disponible sur Internet : http://tel.archives-ouvertes.fr /tel-00455992/fr/

AUXEPAULES L. (2010). Evaluation de la méthode d'appariement ACDC appliquée au diagnostic dans Diagram, un EIAH de la modélisation orientée objet. *Troisièmes rencontres jeunes chercheurs en EIAH* (RJC-EIAH'2010), Lyon, p. 143-144.

BAGHAEI N., MITROVIC A., IRWIN W. (2006). Problem-Solving Support in a Constraint-based Tutor for UML Class Diagrams. *Technology Instruction, Cognition and Learning 2006 (TICL 2006)*, vol. 4, n°2, p. 113-137.

BROWN, A. L. (1987). Metacognition, executive control, self-regulation and other more mysterious mechanisms. *Metacognition, Motivation and Understanding*, F.E Weinert and R.H. Kluwe (eds), chap. 3, p. 65–116. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

CHARROUX B., OSMANI A., THIERRY-MIEG Y. (2005). *UML 2*. Pearson Education France.

DRANIDIS D. (2007). Evaluation of StudentUML : an Educational Tool for Consistent Modelling with UML. *Proceedings of the 2nd Informatics Education Europe Conference* (IEEII), p. 248-256.

EUZENAT J., SHVAIKO P. (2007). *Ontology Matching*, Springer-Verlag, Berlin Heidelberg, 334 p.

FLAVELL J. H. (1976). Metacognitive aspects of problem-solving. *The nature of intelligence*, p. 231-235.

FLAVELL J. H. (1979). Metacognition and cognitive monitoring: a new area of cognitive-developmental inquiry. *American Psychologist*, n° 34, p. 906-911.

FROSCH-WILKE D. (2003). Using UML in Software Requirements Analysis – Experiences from Practical Student Project Work. *Informing Science inSITE*, p. 175-183.

HABRA N., NOBEN K. (2001). Teaching Object Orientation at the Design Level. *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).

LEMEUNIER T. (2000). L'intentionnalité communicative dans le dialogue homme-machine en langue naturelle. Thèse de doctorat en informatique, Université du Maine, Le Mans, France.

MOISAN S., RIGAULT J.-P. (2009). Teaching Object-Oriented Modelling and UML to Various Audiences. *Models in Software Engineering, Workshops and Symposia at MODELS 2009*. Lecture Notes in Computer Science (LNCS) 6002, Springer-Verlag, p. 40-54.

MORITZ S. (2008). *Generating and Evaluating Object-Oriented Designs in an Intelligent Tutoring System*. Philosophiæ Doctor thesis in Computer Science, Lehigh University, 188 p. Disponible sur internet : http://designfirst.cse.lehigh.edu /MoritzDissertation.pdf (consulté le 26 novembre 2010).

MADHAVAN J., BERNSTEIN P., RAHM E. (2001) Generic schema matching with Cupid, *Proceedings of 27th International Conference on Very Large Data Bases (VLDB)*, Rome, Italy, p. 49–58. Disponible sur internet : http://db.cs.washington.edu/papers /CupidTechReport.pdf (consulté le 26 novembre 2010).

OHLSSON S. (1994). Constraint-Based Student Modelling, *Student Modelling : the Key to Individualized Knowledge-based Instruction*, J. E. Greer, G. McCalla (Eds.), Berlin, Springer, p. 167-189.

RAHM E., BERNSTEIN P. A. (2001). A survey of approaches to automatic schema matching. *The international Very Large DataBases Journal (VLDB Journal)*, vol. 10, n°4, p. 334-350.

ROBBINS J. E. (1999). *Cognitive Support Features for Software Development Tools*. Thèse de doctorat. Université de Californie, Irvine.

ROQUES P. (2003). *UML par la pratique*. Ed. Eyrolles.

ROQUES P., VALLEE F. (2007). *UML 2 en action. De l'analyse des besoins à la conception*. Ed. Eyrolles.

SHVAIKO P., EUZENAT J. (2005). A Survey of Schema-based Matching Approaches. *The Journal on Data Semantics*, vol. 4, p. 146-171. Disponible sur internet : http://www.dit.unitn.it/~p2p/RelatedWork/Matching/JoDS-IV-2005_SurveyMatching-SE.pdf (consulté le 26 novembre 2010).

SORLIN S., SOLNON C., JOLION J.-M. (2007). A Generic Graph Distance Measure Based on Multivalent Matchings. Applied Graph Theory in Computer Vision and Pattern Recognition, *Studies in Computational Intelligence*, Springer Berlin, volume 52, p. 151–182.

SURAWEERA P., MITROVIC A. (2004). An Intelligent Tutoring System for Entity Relationship Modelling. *The International Journal of Artificial Intelligence in Education*, vol. 14, n°3-4, p. 375-417.

SURCIN S., CHOPPY C., SERE M.-G. (1995). Analyse didactique d'un cours de Génie Logiciel base sur l'approche orientée objets. *Sciences et Techniques Éducatives*, vol. 2, n° 3, p. 265-286.

Unified Modelling Language™ (1997). *UML® Resource Page*, Object Management Group, http://www.uml.org/ (consulté le 26 novembre 2010).

À propos des auteurs

Mathilde ALONSO est docteur en informatique. Elle a soutenu une thèse en 2009 à l'université du Maine sur le thème de la conception de l'interaction dans un EIAH pour la modélisation orientée objet.

Adresse : LIUM, Université du Maine, Avenue Messiaen, 72085 Le Mans Cedex 9

Courriel : mathilde.alonso@lium.univ-lemans.fr

Ludovic AUXEPAULES est docteur en informatique. Il a soutenu une thèse en 2009 à l'université du Maine sur le thème de l'analyse des diagrammes de l'apprenant dans un EIAH de la modélisation orientée objet.

Adresse : LIUM, Université du Maine, Avenue Messiaen, 72085 Le Mans Cedex 9

Courriel : ludovic.auxepaules@lium.univ-lemans.fr

Dominique PY est professeure en informatique à l'Université du Maine depuis 2003. Elle a soutenu une habilitation à diriger des recherches en 2001, à l'université de Rennes I. Elle effectue actuellement sa recherche au Laboratoire d'Informatique de l'Université du Maine. Ses principaux centres d'intérêt sont la conception des EIAH et la modélisation de l'apprenant.

Adresse : LIUM, Université du Maine, Avenue Messiaen, 72085 Le Mans Cedex 9

Courriel : dominique.py@lium.univ-lemans.fr

Référence de l´article :

Mathilde ALONSO, Ludovic AUXEPAULES, Dominique PY (Laboratoire d&#;Informatique de l&#;Université du Maine, Le Mans), DIAGRAM, un EIAH pour l'initiation à la modélisation orientée objet avec les diagrammes de classe UML, *Revue STICEF*, Volume 17, 2010, ISSN : 1764-7223, mis en ligne le 03/01/2011, http://sticef.org

© Revue Sciences et Technologies de l´Information et de la Communication pour l´Éducation et la Formation, 2010