Programmation sur Exemple et Enseignement assisté par ordinateur de l'algorithmique : le projet MELBA

Nicolas Guibert, Patrick Girard

LISI-ENSMA
BP40109 – 1, avenue Clément Ader
86961 Futuroscope Chasseneuil cedex
{guibert,girard}@ensma.fr

RESUME

L'utilisabilité des langages de programmation et des environnements qui leur sont associés est au centre de multiples études. Celles ci ont débouché sur la création de nombreux systèmes, se donnant pour but de permettre à des non-programmeurs de réaliser des programmes informatiques, sans avoir à apprendre les concepts associés. Dans cet article, nous décrivons l'environnement MELBA (pour Metaphor based Environment to Learn how to Build Algorithms). Celui-ci se donne pour but l'utilisation des techniques de programmation visuelle et « sur exemple » pour supporter l'apprentissage de l'algorithmique, à l'université, à l'aide d'exemples concrets et de métaphores.

MOTS CLES : Programmation sur Exemple, Environnement Informatique pour l'Apprentissage Humain, programmation visuelle.

ABSTRACT

Usabilitity issues in programming languages and associated environments are the topic of multiple studies, which led to the creation of many innovating « programming systems » where the user is the center of the design process, allowing non-programmers to create programs, without understanding the underlaying concepts. In this paper, we describe the MELBA environment (for Metaphor based Environment to Learn how to Build Algorithms). Its purpose is to use advanced visual and example-based programming techniques to support Computer Science (specificly Algorithmic) learning in higher education, by using concrete examples and metaphors.

KEYWORDS : Programming by Demonstration, Computer-Aided Teaching & Learning, Visual Programming.

INTRODUCTION

Les cours d'introduction à la programmation sont tristement remarquables pour leur taux d'échec – ou d'abandon (qui s'échelonne entre 25 et 80% de par le monde, d'après J.Kaasboll[1]). Il faut dire que la programmation telle qu'elle est pratiquée de nos jours soulève naturellement deux types de problèmes qui sont loin d'être triviaux :

- Des problèmes d'utilisabilité des outils, qui sont liés au média (éditeur de texte) employé pour faire communiquer le programmeur et la machine, et à la constitution de la syntaxe des langages en elle-même.
- Des problèmes liés à la nécessité d'abstraire des comportements dynamiques en une seule méthode statique.

Cette double problématique a conduit au développement de nombreux travaux, d'une part d'analyse des processus cognitifs et des facteurs influant sur l'apprentissage de la programmation, d'autre part de production d'environnements destinés aux non-programmeurs ou aux programmeurs débutants (mais dont le but n'était pas l'apprentissage des concepts de la programmation).

Nous nous proposons de combiner ces deux approches, à travers la réalisation d'un environnement informatique pour l'apprentissage humain dédié à l'apprentissage de la programmation impérative, qui utilise les techniques de programmation graphique sur exemple, et qui soit construit à la fois dans un souci d'intégration des théories cognitives liées à la programmation et à l'apprentissage de la programmation et dans un souci d'utilisabilité.

ANALYSES FONDATRICES

L'analyse comparative d'études menées en didactique de l'informatique, telles que celles de Pea [2], nous a amenés à identifier plusieurs grandes classes d'erreurs :

- Une classe d'erreurs « temporelles » (c'est-à-dire venant d'une incompréhension de la dynamique du programme – par exemple dans une itération).
- Une classe d'erreurs « anthropomorphiques » (L'apprenant accorde de façon inconsciente à

l'interpréteur du programme une compréhension contextuelle de son code, dépassant les instructions explicites).

D'autre part, presque toutes les études convergent sur la difficulté des apprenants à bâtir des stratégies des résolutions adaptées à de gros problèmes, quand bien même ils maîtrisent l'écriture des programmes correspondant aux sous-problèmes les composant.

En se référant à une décomposition des connaissances de la programmation commune en didactique de la programmation et connue sous le nom d' « échelle sémiotique » - figure 1-, on peut constater que la plupart des difficultés rapportées ci-dessus sont de nature sémantique (c'est à dire qu'elle sont liées à la compréhension des principes de fonctionnement du langage : itération, programmation modulaire...) ou pragmatique (se rapportant au transfert des connaissances en programmation pour traiter un cas concret).

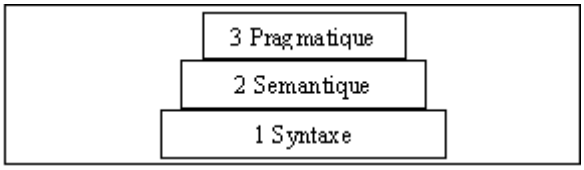

Figure 1 : une décomposition commune des connaissances en programmation : l' « échelle sémiotique ».

Or, le grand nombre d'erreurs de faible importance générée par la couche syntaxique, à cause de nombreux défauts d'utilisabilité [3] (tels que l'écart important entre les concepts du système et de l'utilisateur ou la surcharge de la charge mémoire induite par l'inférence du comportement du système, aggravés par un feedback peu efficace, et parfois même par des rapports d'erreurs hermétiques) nous paraît elle non seulement être une cause importante de frustration, et donc d'échec, mais également focaliser l'attention des étudiants, -et l'assistance de leur encadrant- sur ces problèmes syntaxiques, au détriment des couches supérieures.

Cette prise en compte tardive de l'usage réel des compétences de programmation semble de plus s'opposer en partie aux heuristiques pour l'enseignement de tâches informatiques, telles que les définit Caroll [4], et qui recommandent, en particulier :

- que chaque tâche d'enseignement débouche sur une connaissance directement applicable par l'apprenant.
- que les cours soient liés à l'utilisation « réelle » de l'outil ou du langage.

INTRODUCTION A MELBA

Nous nous proposons de construire un environnement d'apprentissage de l'algorithmique à l'usage des étudiants en début de cursus universitaire ou plus généralement de cursus d'enseignement supérieur. Celui ci prend en compte les problèmes cités auparavant, et fournit ainsi une utilisabilité accrue grâce à un écho immédiat (simulation de l'algorithme pendant l'édition), et une recherche d'erreur facilitée par un contexte graphique mis en place à l'aide d'une métaphore.

L'environnement se décompose en deux parties qui sont « synchronisées » : l'espace de manipulation de l'algorithme, d'une part, et l'espace de manipulation du contexte, d'autre part.

L'espace de manipulation de l'algorithme permet à l'étudiant de construire son algorithme à partir d'un formalisme graphique (voir figure 2) préalablement vu en cours.

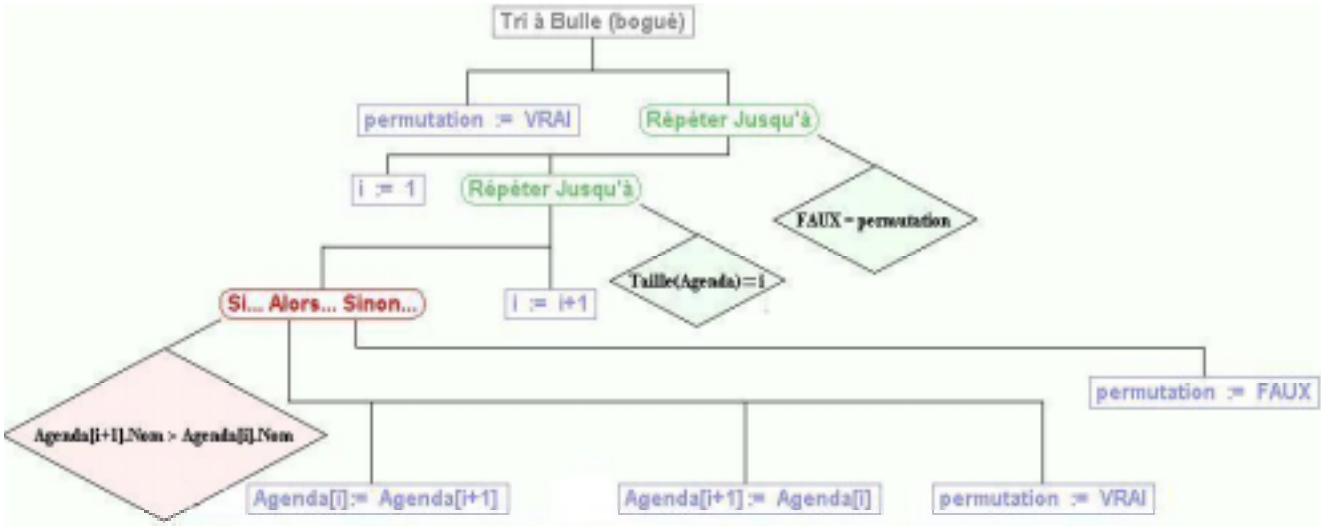

Figure 2 : Edition d'un algorithme de tri par bulle (erroné)

Le choix d'un formalisme graphique a été fait conformément à plusieurs impératifs : premièrement dissocier l'algorithmique de la connaissance d'un langage donné, et ainsi favoriser l'émergence de savoir-faire génériques et réutilisables, et deuxièmement faire abstraction d'un certain nombre de problèmes syntaxiques, au bénéfice des difficultés d'ordre sémantique et pragmatique.

L'espace de manipulation du contexte représente, à l'aide d'une métaphore, les données (variables, constantes, paramètres…) de l'algorithme. Il est synchronisé sur l'espace de manipulation de l'algorithme et fournit donc à chaque instant un écho graphique immédiat par rapport à la manipulation du graphe. En effet, après chaque édition, l'algorithme est réévalué jusqu'à l'instruction éditée. D'autre part il est possible de rejouer entièrement l'algorithme une fois le graphe complété. Ses différents objets sont directement manipulables, et leur manipulation entraîne une modification de l'algorithme (création, affectation de variables, appels de procédures…).

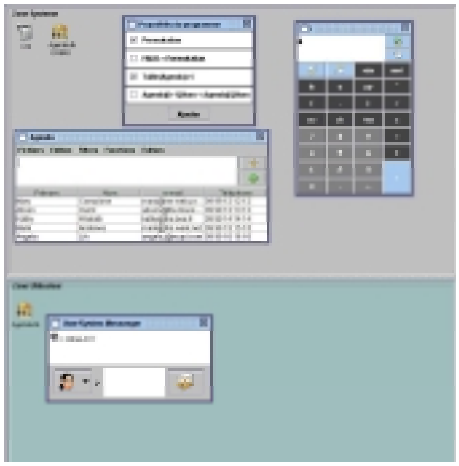

Figure 3 : L'espace du contexte associé à l'algorithme.

Il se décompose en deux sous espaces : d'une part le contexte interne du programme (zone système), d'autre part le contexte externe, c'est à dire les paramètres et les entrées | sorties avec l'utilisateur (zone utilisateur).

ECHO IMMEDIAT ET CONTEXTE GRAPHIQUE

Alors que la possibilité de Programmation sur exemple à pour but de fournir une aide à l'étudiant lorsqu'il ne voit pas quelles instructions utiliser, l'utilisation de l'exemple en cours d'édition ou à posteriori, en tant qu'écho, lui permet de détecter immédiatement un certain nombre d'erreurs de sémantique.

Ainsi, si on chargeait l'exemple de tri erroné, l'écho nous permettrait de détecter immédiatement deux erreurs de nature sémantique : l'une lors de l'échange des deux lignes (figure 4) et l'autre dans le …sinon.

On peut constater que ces deux erreurs proviennent de deux causes différentes, ainsi le bug lors de l'échange des lignes se rangerait dans les deux catégories « anthropomorphique » et « temporelle », car « tu mets la première à la place de la seconde, et vice-versa » correspond bien à ce que l'on expliquerait à un interpréteur humain et car cela suppose une résolution simultanée et non séquentielle. La seconde erreur est elle clairement de type temporelle : l'étudiant ne comprend pas que mettre permutation à faux le fera sortir immédiatement de la boucle (mais le constate de suite grâce à l'écho immédiat fourni par le contexte). Les deux types d'erreur de conception peuvent ainsi être détectés facilement grâce à l'écho immédiat et à la métaphore qui permettent une visualisation de la dynamique du programme en cours de conception.

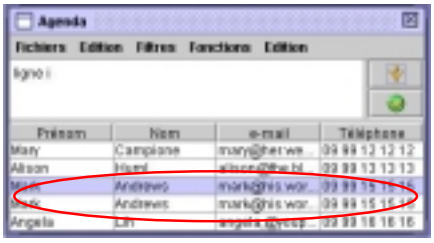

Figure 4 : écho de l'exécution de « Agenda[i]:=Agenda [i+1] »

CONTEXTE(S) « INTERNE » ET « EXTERNE ».

La distinction entre la partie « interne » et « externe » du contexte permet de désigner graphiquement les paramètres de l'algorithme (dans l'exemple du tri, il s'agit du tableau Agenda.tb qui est passé par recopie dans l'espace interne du contexte). D'autre part elle nous permet de dissocier complètement la connaissance que l'interpréteur a des variables de celle de l'utilisateur du programme.

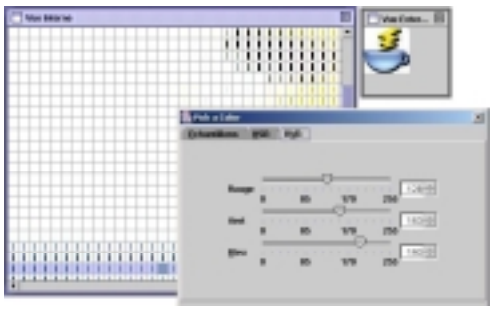

Figure 5: un élément présent dans les deux parties du contexte : externe (l'image) et interne (un tableau de couleurs)

Par exemple dans la figure 5, on peut voir que contrairement à l'utilisateur qui interprète une image par son information visuelle, le programme l'interprète comme un tableau de couleurs.

Il sera bien sûr possible d'attacher à chacun de ces objets des parties contrôle prenant en charge la programmation

par l'exemple : cela correspond à la création de nouveaux types et de nouvelles bibliothèques de fonctions dans des langages et environnements « classiques ». Ce caractère « ouvert » discerne MELBA de la plupart des systèmes de PsE, qui ne sont pas aussi extensibles.

DISCUSSIONS SUR LA METAPHORE

Le choix de la métaphore du bureau pour le contexte graphique s'est fait en accord avec plusieurs critères. Tout d'abord, un critère de « transparence ». Nous entendons par là que la métaphore du bureau est supposée connue de tous les utilisateurs, fussent-ils occasionnels, d'ordinateurs. Il paraîtra naturel au public visé de manipuler des nombres via une calculatrice ou des tableaux via des feuilles de calculs. A contrario, une métaphore plus sophistiquée, telle celle du jeu de construction dans ToonTalk [5], aurait pu nécessiter un effort d'apprentissage du fonctionnement de l'espace du contexte, ce qui risquait de détourner l'attention de l'apprenant de l'algorithme en lui-même …

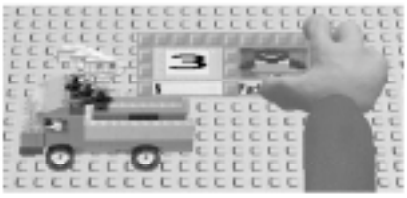

Figure 6 : La métaphore du jeu de construction dans ToonTalk

Ensuite, il fallait maintenir une cohérence entre les actions possibles en programmation impérative et les interactions autorisées dans l'espace d'écho graphique et de PsE ; ceci a été possible dans la métaphore choisie, en identifiant naturellement l'action d'affectation d'une constante à une variable à l'action de saisie, l'affectation de variable à variable au copier|coller, et les procédures de plus haut niveau à l'utilisatation d'éléments de barres d'outils ou de menus. Les différentes fonctions peuvent être regroupées en bibliothèques au sein de fenêtres d'outils indépendants ou d'édition de document.

Enfin, l'usage de la métaphore du bureau nous permet d'assurer une plasticité et une adaptabilité importante de l'interface, car le contenu des fenêtres dans la métaphore du bureau est peu contraint. Cette adaptabilité, liée à la métaphore, est ici mise au service de l'enseignant qui conçoit l'exercice, et qui peut de cette façon mettre à la disposition de l'apprenant des données présentées et contrôlées d'une façon compatible à la démarche de l'exercice. Il aurait ainsi pu choisir dans l'exemple de la figure 5 un enregistrement R, V & B ou un unique entier pour représenter la couleur, selon les buts qu'il se serait fixé. Il aurait aussi pu ajouter au composant « interne » une série de fonctions pour factoriser des traitements supposés connus ou pour permettre une approche en décomposition au niveau de la compréhension de la tâche (c'est-à-dire s'attacher à décrire le déroulement de la tâche avec des fonctions de haut niveau avant de décrire ces fonctions à l'aide de primitives de plus bas niveau, et ainsi de suite …)

Il peut également grâce à l'adaptabilité de l'interface construire des documents et des exercices fortement liés à la filière de l'étudiant (informatique pour la biologie, pour la physique ou la chimie …), par exemple un document d'entrée peut être composé de tableaux et de courbes représentant les résultats d'une expérience de physique.

CONCLUSION ET PERSPECTIVES

Cet article décrit l'environnement d'aide à l'apprentissage de l'algorithmique MELBA, qui est destiné aux étudiants suivant des cours d'initiation à l'algorithmique dans l'enseignement supérieur. Il est caractérisé par une description graphique du programme couplé à un espace de programmation sur exemple fournissant un écho graphique immédiat à la manipulation de l'algorithme, dans le but de permettre une détection immédiate et une compréhension facilitée des erreurs sémantiques dans la conception de programmes.

La prochaine étape consiste à mener une étude sur l'usage de MELBA en conditions réelles, dans le but de valider les idées développées ici en analysant l'impact de cet environnement sur des étudiants en apprentissage de l'algorithmique dans l'enseignement supérieur. Puis nous nous attacherons à développer un environnement de PsE dédié à l'enseignant permettant de concevoir interactivement des exercices. Enfin nous étudierons les modalités d'extensions de MELBA à d'autres contextes que les travaux dirigés, tels que des projets en groupe ou des exercices d'auto-évaluation en ligne.

BIBLIOGRAPHIE

1. Kaasboll, J., Learning Programming, . 2002, University of Oslo.
2. Pea, R.D., Language-Independent Conceptual "Bugs" in Novice Programming. Journal of Educational Computing Research, 1986. 2(1): p. 25-36.
3. Nielsen, Usability Engineering. ISBN 0-12-518405-0 ed. 1993: Academic Press.
4. Caroll, J.M., The NurnBerg Funnel. 1990, Cambridge: MIT Press.
5. Kahn, K., How Any Program Can Be Created by Working with Examples, in Your Wish is My Command, H. Lieberman, Editor. 2001. p. 21-44.