Visualisation dynamique de programmes, artEoz : l'outil qui manquait

Martine Gautier, Brigitte Wrobel-Dautcourt

Visualisation dynamique de programmes, *artEoz :* l'outil qui manquait

Martine GAUTIER, Brigitte WROBEL-DAUTCOURT
martine.gautier@loria.fr , brigitte.wrobel-dautcourt@loria.fr

LORIA Laboratoire lorrain de recherche en informatique et applications
UMR 7503 - Campus Scientifique – BP 239 – 54506 VANDOEUVRE-lès-Nancy Cedex

Résumé. artEoz est un logiciel de visualisation dynamique de programmes. Il est né de notre expérience pédagogique de l'enseignement de la programmation objet. La compréhension de l'effet des instructions d'un programme passe par la construction d'une représentation mentale de ce qui se passe en mémoire centrale lors de l'exécution d'un programme instruction par instruction. Le travail de l'enseignant est de guider l'apprenant dans la construction de cette représentation de sorte qu'il puisse s'y appuyer en toutes circonstances. L'objectif principal du logiciel **artEoz** est de mettre en images les paradigmes de la programmation objet, allant de la simple déclaration de variable à l'instanciation et à l'appel de fonctions, tout en restant beaucoup plus facile d'accès qu'un outil classique de débogage. En concurrence directe avec de nombreux autres logiciels de visualisation de programmes, nous présentons les motivations de développement de notre logiciel, notre démarche de conception, nos objectifs pédagogiques et le retour des premiers utilisateurs.

Mots-clés: Visualisation dynamique de programmes, programmation objet, pédagogie, schéma mémoire, plateforme logicielle.

Introduction

L'apprentissage de la programmation objet passe par la compréhension de paradigmes spécifiques que sont par exemple la création d'objets, la notion d'appel de fonction avec receveur ou encore la liaison dynamique. Ces paradigmes se retrouvent dans tous les langages objets sous des formes voisines. Un programmeur est efficace s'il sait jongler avec ces mécanismes et ainsi passer d'un langage à un autre. Se contenter d'apprendre la syntaxe d'un langage est très largement insuffisant.

Pour atteindre cet objectif, l'enseignant est depuis longtemps à la recherche de moyens pédagogiques pour encourager cet apprentissage (Bataller Mascarell, 2011). Parmi ceux-ci, les logiciels de visualisation d'exécution de programmes ont déjà largement fait preuve de leur utilité. Il existe de nombreuses études sur les bénéfices de l'utilisation de tels logiciels dans un cadre éducatif (Ben Ari, 2001). En effet, la visualisation et l'animation de programmes augmentent la motivation des élèves (Urquiza Fuentes & Velázquez Iturbide, 2013), aident à l'inévitable recherche de bugs à l'image de (Cross II, Hendrix & Barowski, 2011) qui proposent une démarche permettant de s'appuyer sur la visualisation pour garantir le respect des tests unitaires.

Ainsi, on trouve facilement des outils[1] pour visualiser l'évolution de programmes, mais curieusement, leur utilisation n'est pas systématique. Les raisons en sont multiples. Le principal obstacle est l'effort demandé aux enseignants et aux apprenants pour prendre en main et maîtriser les outils proposés avant que ceux-ci ne remplissent leur rôle de support à la pédagogie (Naps et al., 2002). Cet effort est généralement rédhibitoire. La seconde raison est purement d'ordre pédagogique. Pour qu'une équipe pédagogique puisse s'appuyer sur un tel outil, il est indispensable que la démarche des enseignants soit en adéquation avec la visualisation proposée. En particulier, le niveau d'abstraction des schémas proposés doit être en cohérence avec le niveau des étudiants à qui il s'adresse. Pour susciter la curiosité des utilisateurs, qu'ils soient enseignants ou étudiants, il est important que l'outil logiciel soit très simple à utiliser tout en offrant un large éventail de fonctionnalités. La troisième raison concerne

1 http://algoviz.org/avcatalog

l'activité des apprenants. Il ne suffit pas d'avoir un outil qui propose des animations, il faut également que l'élève devienne acteur. Plus l'élève est impliqué dans les animations, plus il apprend.

Conscientes de ces difficultés, nous proposons un logiciel de visualisation de programmes, **artEoz**, conçu et développé par une équipe pédagogique, forte d'une expérience de l'enseignement de la programmation objet depuis 25 ans. Ce logiciel se veut intuitif à prendre en main et à utiliser, l'effort d'apprentissage doit se focaliser sur les objectifs pédagogiques et non sur l'utilisation de l'outil. De plus, pour impliquer et faire participer les élèves, **artEoz** est le noyau d'une plate-forme pédagogique destinée à inclure des scénarios d'apprentissage.

Contexte

Motivation

L'enseignement de l'informatique propose généralement une solide formation en algorithmique et programmation dans un langage évolué, le plus souvent orienté objet. Récemment encore, cette formation était complétée par un enseignement en langage assembleur, très proche de la machine. Cependant, au vu du temps imparti à la formation des étudiants et devant l'ampleur et l'accroissement des notions fondamentales à enseigner, l'enseignement de la programmation assembleur, souvent qualifié d'obsolète aujourd'hui, a la plupart du temps disparu au profit de l'enseignement de concepts de plus haut niveau, plus proches de la réalité quotidienne du développeur. Certains arguent que les concepts des langages de programmation évolués suffisent, qu'il est inutile de descendre à un niveau plus bas, puisque les apprenants ont l'habitude des concepts abstraits directement issus des mathématiques. D'autres avancent comme argument la comparaison avec la conduite d'une voiture qui ne nécessite aucune connaissance sur le fonctionnement d'un moteur à quatre temps. La suppression de l'enseignement d'un langage très bas-niveau n'est cependant pas sans conséquence sur l'enseignement de la programmation de haut niveau, tout comme les notions de mécanique sont utiles au pilote de course, si on garde la même comparaison.

La disparition de l'enseignement de l'assembleur est inéluctable. Toutefois, la connaissance du mécanisme de la programmation très proche de la machine permet de donner un sens à la notion de programmation en langage évolué. Une affectation, par exemple, est tout de même un concept assez abstrait, qui prend tout son sens (dans tous les sens du terme ...), lorsqu'on fait le rapprochement avec un transfert de mémoire à registre ou de registre à registre. De même, un appel de fonction récursive avec passage de paramètre ne revêt plus aucun caractère magique lorsqu'on comprend l'évolution de la pile à l'exécution.

L'écriture de programmes assembleur favorise également l'intégration des notions simples d'efficacité : dans un programme écrit dans un langage évolué, chaque instruction donne faussement l'impression d'avoir une efficacité comparable à sa voisine, en temps d'exécution et en place mémoire requise. Dans un programme assembleur, on apprend vite à faire la différence entre les temps d'exécution d'une addition et d'une multiplication et la réservation d'espace mémoire inutile est évidente.

La disparition de l'apprentissage d'un langage très bas niveau et la diminution de l'enseignement consacré au fonctionnement d'un ordinateur a des conséquences sur la compréhension et l'agilité dans la programmation en langage de haut niveau. Les niveaux d'abstraction successifs s'éloignent de plus en plus de l'octet. Or, la formation de base des informaticiens passe par un apprentissage solide des notions de programmation. Pour notre société en pleine évolution numérique, il est essentiel de former des programmeurs de haut-niveau et non des "programmeurs-bidouilleurs", qui travaillent sans comprendre, par d'incessants copier/coller. De plus, malgré la taille *gigatesque* des mémoires et la capacité de calcul des processeurs actuels, il existe un domaine en plein essor dans lequel les performances sont importantes : les logiciels embarqués.

Tous les enseignants confrontés aux problèmes rencontrés par des débutants savent que la compréhension de l'effet des instructions d'un programme passe par la construction d'une

représentation mentale de ce qui se passe en mémoire centrale lors de l'exécution, instruction par instruction. Alors que cette représentation peut se faire aisément dans des cas simples comme l'affectation, elle devient beaucoup plus difficile à imaginer lorsqu'on travaille sur des constructions plus complexes comme la création d'objets en programmation objet ou l'appel de fonction avec passage de paramètres de types différents.

Le travail de l'enseignant est de guider l'apprenant dans la construction de cette représentation, de sorte qu'il puisse s'y appuyer en toutes circonstances. Les débogueurs fournissent à ce titre une multitude d'informations sur l'état de la mémoire, mais un débutant est d'une part, rebuté par l'effort à fournir pour utiliser un tel outil, et d'autre part, il est complètement perdu dans cette masse de données visualisées. De plus, il est inutile d'avoir une représentation exacte de toutes les informations présentes en mémoire ; une abstraction simplifiée de la réalité suffit, du moment que le modèle utilisé est cohérent et complet.

Démarche

Notre démarche a débuté par l'identification d'une forme de représentation, c'est-à-dire un modèle de données, qui soit à la fois cohérent et complet, facilement extensible pour intégrer de nouvelles constructions et également visuellement le plus simple possible. Pour être applicable à la programmation objet, le modèle intègre différents types de données : les variables, les objets, la pile des environnements créés lors des appels de fonctions. Une variable est complètement définie par son nom et son contenu ; un objet est défini par son type dynamique et ses attributs, qui sont eux-mêmes des variables. Le contenu d'une variable est, selon son type, une valeur (entier, réel, caractère, booléen, ...) ou l'adresse d'un objet. Un environnement d'appel de fonction est défini par l'ensemble des variables concernées, c'est-à-dire le receveur de l'appel et les variables paramètres.

Le modèle change d'état lors de l'exécution d'une instruction : une variable change de valeur, une nouvelle variable ou un nouvel objet apparaît ou disparaît. Cette démarche pédagogique est depuis longtemps utilisée par les enseignants au tableau et à la craie. Le dessin (figure 1) repris par un étudiant dans ses notes de cours schématise les variables et les objets d'un programme simple de quelques lignes (le texte du programme n'a que peu d'intérêt ici). Ce modèle change d'état si le programme est complété par une nouvelle affectation ou création..

Force est de constater l'ingratitude de la tâche. Le schéma devient assez vite illisible ; il est impossible de conserver une trace de l'évolution du schéma, qui a pourtant plus d'importance que l'état final.

C'est la dynamique de construction du schéma qui est fondamentale pour la compréhension et non le schéma en lui-même. L'exercice pédagogique est très constructif, mais reste rébarbatif, de par sa construction manuelle. L'usage de la craie et de l'éponge ne suffit pas.

Pour tirer pleinement profit de l'apport de cette démarche, nous l'avons récemment accompagnée de la réalisation d'une machine virtuelle, **artEoz**, qui exécute un programme et visualise le modèle de données correspondant, avec différents niveaux de détails. Cette dernière possibilité permet de se focaliser sur certains aspects au détriment d'autres, qui ne sont pas affichés.

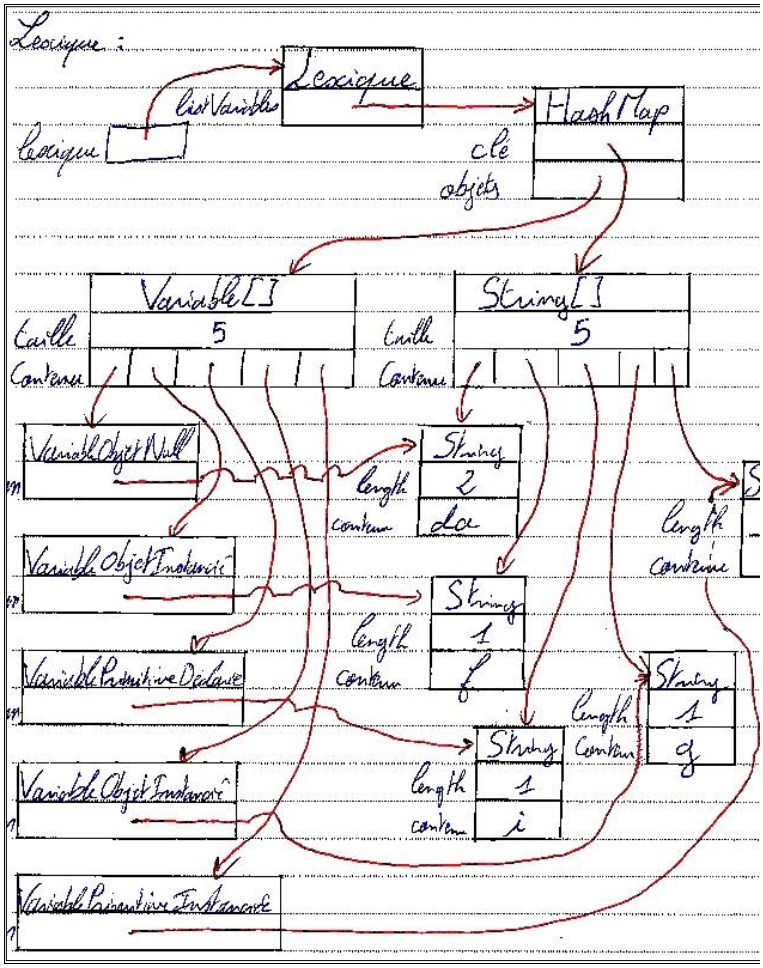


Figure 1. Représentation schématique des variables et objets en mémoire après l'exécution de quelques instructions de programme.

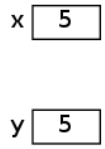
Objectifs pédagogiques

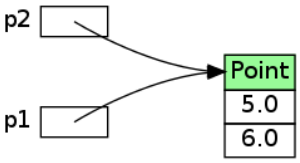
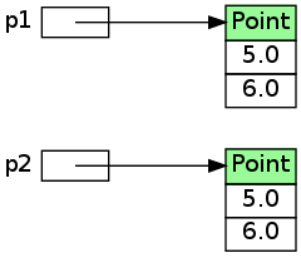
L'objectif principal du logiciel **artEoz** est de fournir un support visuel permettant d'appréhender la dynamique de l'exécution d'un programme en langage objet comme Java. Il n'est pas question d'apprendre la syntaxe d'un langage, l'objectif est de mettre en image des paradigmes tels que la déclaration, l'affectation, la création d'objets ou l'appel de fonction.

> ***Exemple de l'affectation de deux variables de type primitif et de l'affectation de deux variables objets.*** L'affectation d'une variable à une autre variable, que ces variables soient de type primitif ou de type objet, est exécutée par le même mécanisme : on prend le contenu de la variable "source" et on le place dans la variable "destination". Cela paraît généralement clair pour tout le monde. Cependant, l'illustration du résultat de l'affectation par un schéma montre de façon évidente la différence en fonction du type de la variable.
>
> Le tableau 1 illustre ce propos : le code assembleur (colonne 2) correspondant à une affectation de variable (colonne 1) est identique quelque soit le type de la variable affectée, entier ou objet.
>
> Le schéma mémoire (colonne 3) est une mise en image de ce processus. La colonne 4 détaille les problèmes rencontrés par les apprenants.

Tableau 1. Différence entre l'affectation de deux variables de type primitif et de deux variables de type objet. Illustration de l'utilité des schémas mémoire.

Code source	Assembleur	Schéma mémoire	Difficultés de compréhension

`int x, y ;`
`x = 5 ;`
`y = x ;`
	`A0 RES 4`
	`LDW R1, 5`
	`STW R1, @A0+0`
	`LDW R1, @A0+0`
	`STW R1, @A0+2`
			Les élèves n'ont généralement aucune difficulté pour réaliser cette affectation et comprendre le mécanisme sous-jacent.

`Point p1, p2 ;`
`p1 = new Point(5,6) ;`
`p2 = p1 ;`
	`A0 RES 4`
	`LDW R1, ...`
	`STW R1, @A0+0`
	`LDW R1, @A0+0`
	`STW R1, @A0+2`
		
		 
			Sans assembleur et sans représentation de la mémoire, les élèves proposent souvent une duplication de l'instance de la classe, ce qui est faux.

Si on supprime l'apprentissage de l'assembleur, il faut garder une représentation symbolique de ce qui se passe en mémoire afin de ne pas arriver à une mauvaise compréhension de ce que fait réellement une affectation. La construction manuelle d'un dessin est fastidieuse ; son automatisation ouvre de nouvelles perspectives. Le logiciel **artEoz** permet à un apprenant d'étudier la dynamique de l'exécution du programme, de se focaliser sur un point précis, de l'exécuter instruction par instruction, au besoin de revenir en arrière. **artEoz** favorise l'apprentissage en donnant la possibilité de faire des tests à l'infini, sans le côté rébarbatif du dessin manuel ; il ouvre des possibilités de construction de scénarios pédagogiques mettant en lumière des constructions particulières ; il ouvre également des possibilités d'évaluation des connaissances acquises par l'apprenant.

État de l'art

Notre logiciel **artEoz** s'intègre dans le domaine des outils de visualisation de programmes. Ce domaine est vaste, actif depuis plus de trente ans, et pour lequel a déjà été proposée une multitude de logiciels. Cependant, seule une petite partie d'entre eux sont des outils dédiés à la pédagogie et peu d'entre eux sont encore actifs et distribués aujourd'hui. On pourra lire un état de l'art détaillé dans la thèse de J. Sorva (Sorva, 2012). Pour le lecteur intéressé, un catalogue de logiciels à télécharger se trouve sur le site http://algoviz.org/avcatalog.

Parmi les logiciels de visualisation dynamique de programmes orientés objet, citons **BlueJ** (M. Kölling, 2012), apparu vers 1996, **JEliot3** en 2003 (Moreno, Myller, Sutinen, & Ben Ari, 2004; Ben Ari et al., 2011) plus récemment **UUhistle** (Sorva & Sirkiä, 2010) et **Online Python Tutor** de 2010 (Guo, 2013) dont l'ergonomie, la modélisation de l'exécution d'un programme et les objectifs s'apparentent le plus à ceux d'**artEoz**.

Présentation de la plateforme artEoz

L'interface graphique du logiciel **artEoz** est tout à fait classique (figure 2). Nous l'avons voulue simple et ergonomique, dans la continuité de la présentation des différentes plateformes de développement (eclipse, netbeans) ou de débogueurs. L'interface se compose de trois fenêtres principales : l'une, à gauche, pour saisir des instructions, une autre, la plus grande, pour visualiser les schémas mémoire et une troisième pour afficher les messages d'erreur et jouer le rôle de la sortie standard des instructions d'écriture figurant dans le code de l'utilisateur. L'interface graphique contient bien évidemment des menus et des icônes permettant de paramétrer le schéma, par exemple de visualiser les objets morts, les champs hérités, la pile à l'exécution, de gérer le mode pas à pas ... D'autres captures d'écran et des films d'animation se trouvent sur le site http://arteoz.loria.fr, ils illustrent le tutoriel du logiciel.

Dans l'exemple de la figure 2, les instructions écrites dans la fenêtre de code déclarent et affectent des variables de type primitif, des variables de type objet, sur la base des classes *Point*, *Segment* et *Triangle*. Ces trois classes sont propres à l'utilisateur, le logiciel intègrant la possibilité d'importer ses propres ressources. Le schéma mémoire visualise les valeurs des différentes variables et identifie trois objets morts (en gris) devenus inaccessibles après l'exécution de la dernière instruction.

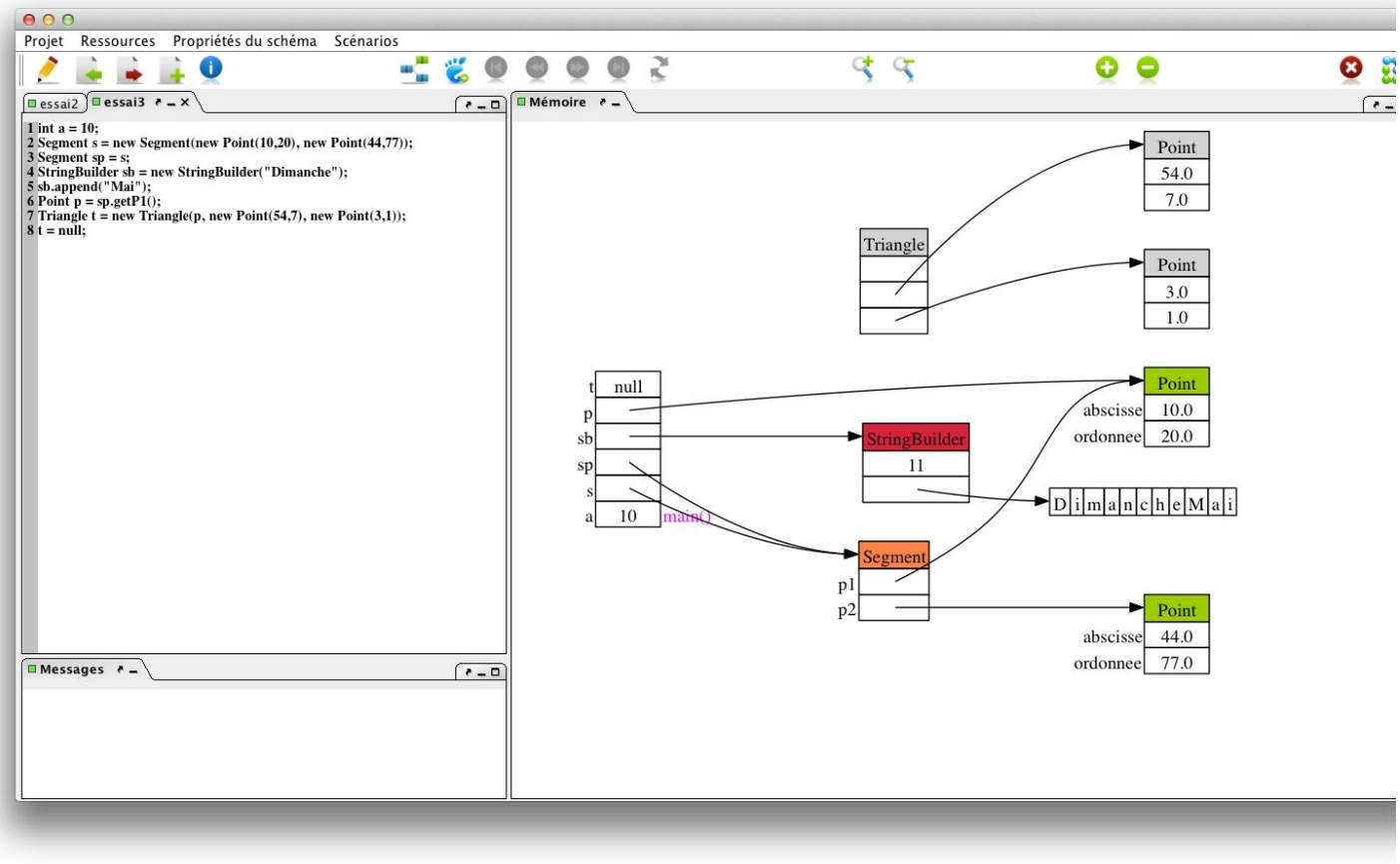


Figure 2 : L'interface graphique du logiciel artEoz.

Évaluation

Public concerné

Le public visé par cette plateforme est très large. Il s'adresse d'une part à des novices en programmation, qu'ils soient lycéens, étudiants ou professionnels. D'autre part, c'est un outil pédagogique mis à la disposition des enseignants pour illustrer les concepts de base de la programmation. L'expérience montre que, sans l'avoir initialement prévu, cet outil sert aussi au programmeur chevronné qui a besoin de se faire une idée précise du comportement d'une séquence d'instructions. Par exemple, nous avons particulièrement apprécié de visualiser l'évolution d'une table de hachage ou l'insertion dans une liste chaînée.

Utilisation libre

L'utilisateur écrit des séquences d'instructions et étudie le schéma mémoire correspondant. Ce type d'utilisation sommaire n'a de chance d'être bénéfique que si l'utilisateur a un objectif précis. Il s'adresse donc à l'enseignant qui prépare son cours, au programmeur averti qui cherche à comprendre l'implantation d'une classe particulière de la bibliothèque. Elle est également utile au programmeur qui doit trouver une erreur dans une séquence d'instructions bien identifiées.

Scénarios d'exécution

L'utilisateur utilise/modifie/affine/transforme des séquences d'instructions toutes faites qui définissent des scénarios prédéfinis. Un scénario est construit pour mettre en évidence le fonctionnement spécifique d'une instruction ou d'une séquence d'instructions, une particularité du langage, l'utilisation d'une classe de la bibliothèque.

L'exemple de scénario de la figure 3 illustre le mécanisme de la liaison dynamique propre à la programmation objet. La séquence d'instructions crée une expression, somme de deux nombres. Elle instancie pour cela les classes **Somme** et **Nombre**, sous-classes de la classe **Expression**. Comme l'indique le diagramme de classes de la figure 3, ces différentes classes implantent la fonction *valeur* de façon différente.

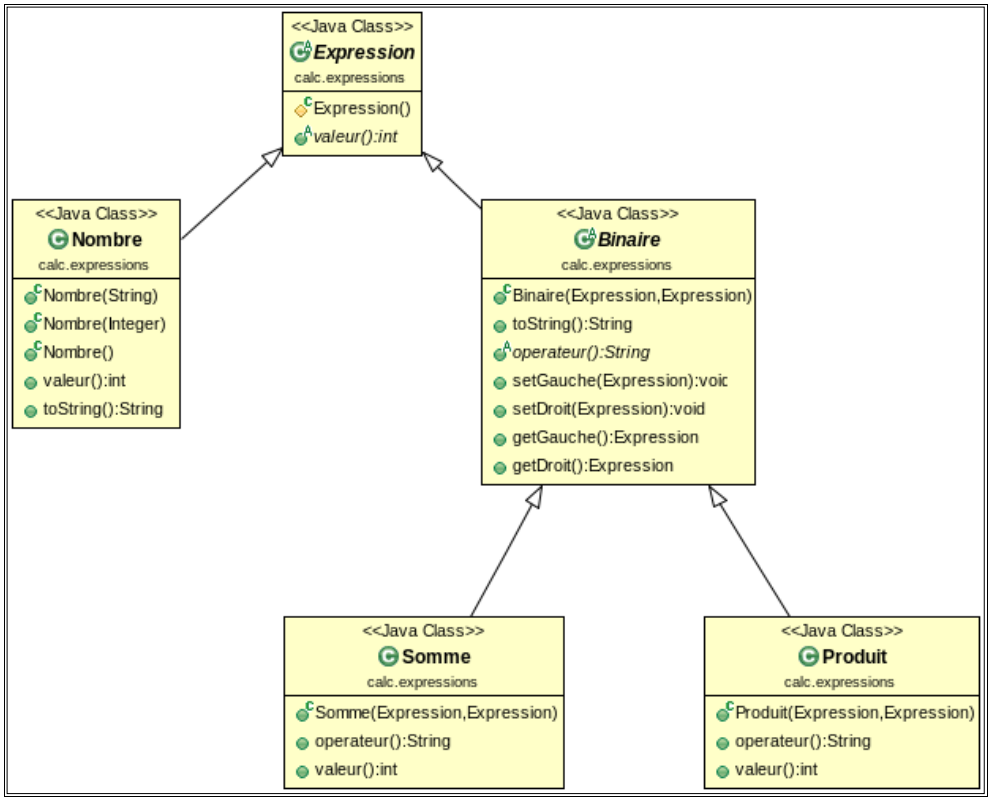


**Figure 3 : Graphe d'héritage : les classes Nombre et Binaire héritent de la classe Expression.
Les classes Somme et Produit héritent de la classe Binaire.
Les classes Nombre, Somme et Produit redéfinissent la fonction *valeur*.**

L'appel de la fonction *valeur* de la ligne 5 (figure 4) de la séquence d'instructions est soumis à la liaison dynamique. Dans la pile visualisée sur le schéma mémoire, le receveur de l'appel (désigné par *this*) est une instance de la classe **Somme**, c'est donc la fonction *valeur* de cette classe qui va être exécutée. Cela démystifie complètement le côté magique de la liaison dynamique, puisqu'on imagine aisément que la machine virtuelle connaissant le type dynamique de l'objet est capable de retrouver la fonction adéquate.

On peut compléter ce scénario par une séquence d'instructions qui crée une liste d'expressions sur le même modèle et calcule la valeur de chacune d'elles (figure 5). Le receveur est désormais de type **Produit** ; c'est la fonction *valeur* de cette classe qui s'exécute. L'utilisateur a ensuite la possibilité de transformer à loisir cette séquence d'instructions pour s'approprier ce mécanisme.

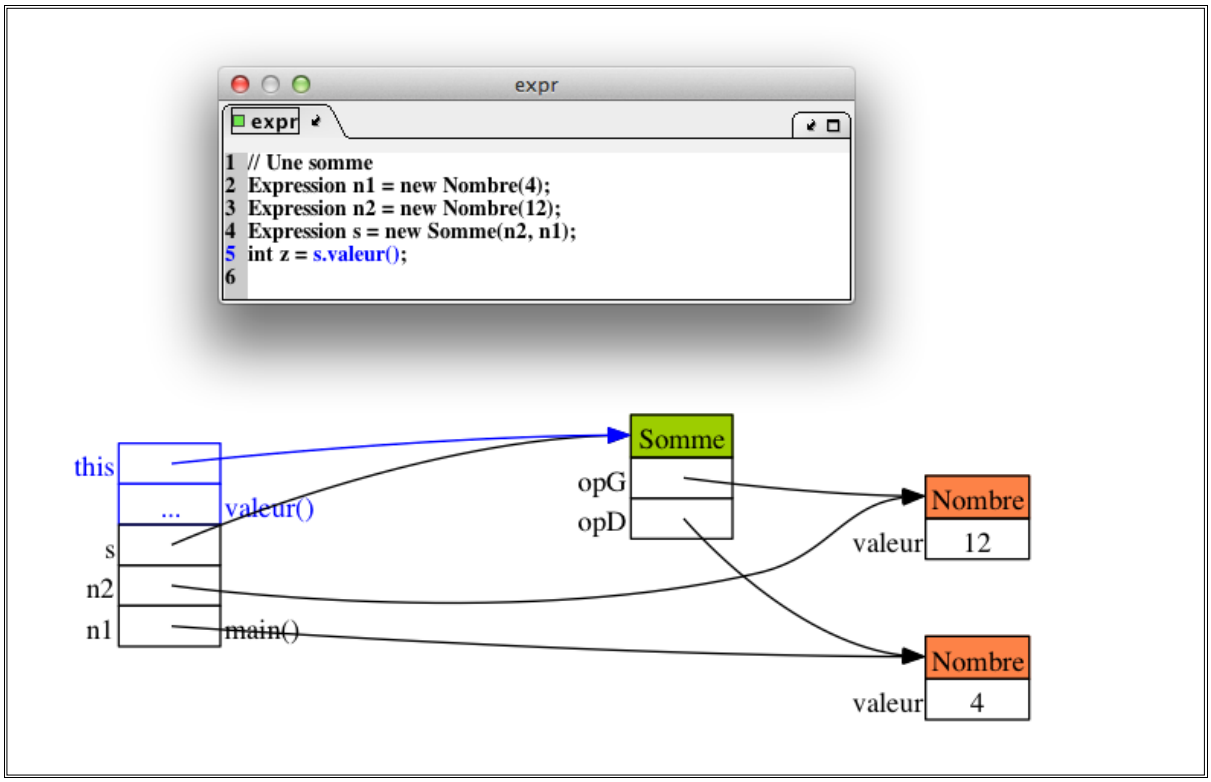
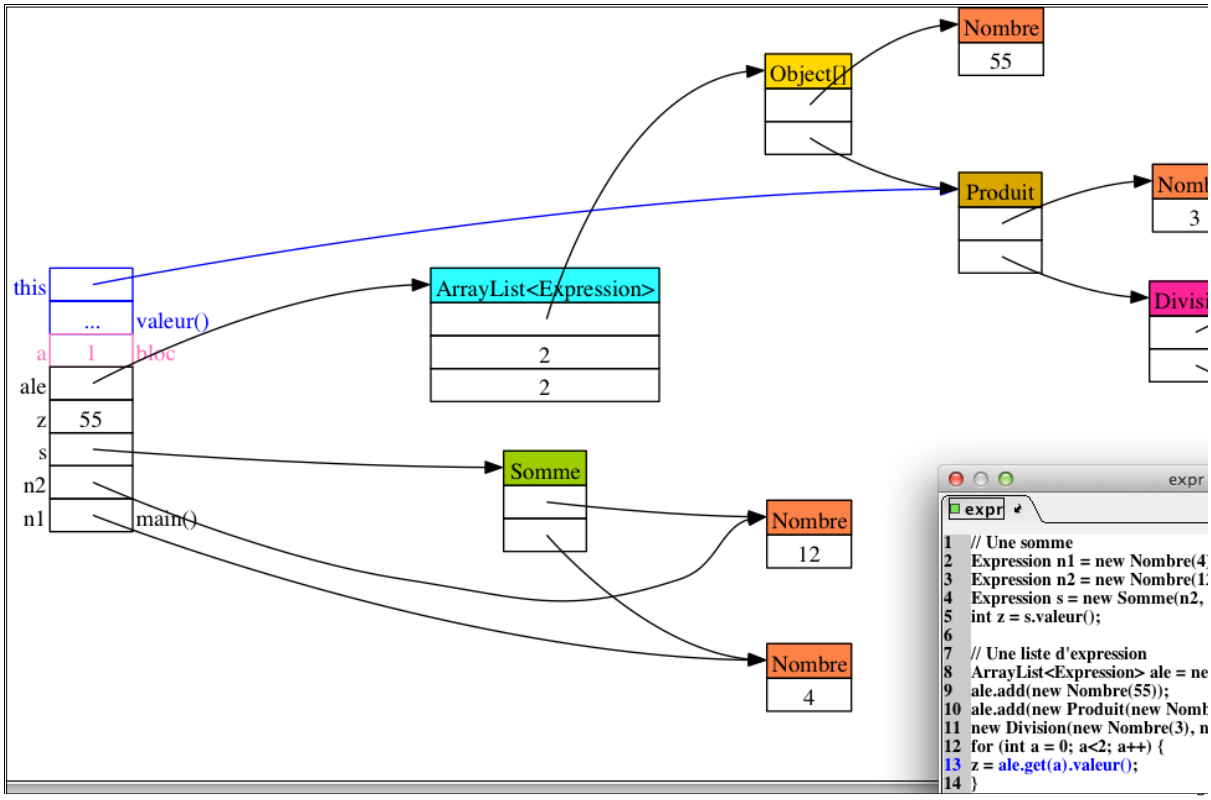


Figure 4 : Illustration du mécanisme de la liaison dynamique ;
étape 1 : appel de la fonction *valeur* sur un objet de type dynamique Somme.



Retours d'expérience

Nous avons mené plusieurs expériences avec différents groupes d'étudiants, essentiellement en licence d'informatique 2ème année, ayant un petit bagage en algorithmique et programmation C. Ils accueillent **artEoz** avec enthousiasme ; sa prise en main est simple et intuitive. Des idées de séquences d'instructions à tester fusent de toutes parts. Curieusement, les schémas mémoire sont à leurs yeux plus crédibles que ceux que l'on fait à la main, même s'ils sont en tous points identiques (en dehors du design). La mise en évidence des objets morts retient leur attention et permet d'amorcer des discussions autour de la gestion mémoire et des ramasse-miettes.

L'évaluation d'**artEoz** est pour l'instant empirique car nous terminons son développement. Il est entre les mains d'enseignants chargés de Tds dans différentes formations et nous allons profiter de cette nouvelle rentrée pour l'utiliser en séances encadrées avec des débutants en programmation.

En parallèle, nous travaillons à la mise en place d'une évaluation explicite et systématique qui nous permettra de mesurer l'apport pédagogique d'**artEoz**. De nombreux autres logiciels de visualisation de programmes ont déjà été évalués et nous bénéficierons ainsi d'une méthodologie d'évaluation (Sensalire, Ogao & Telea, 2009).

Perspectives

Outil de dessin

À l'heure actuelle, un outil de dessin est en cours de développement avant d'être intégré à la plateforme **artEoz**. Cet outil permet à l'utilisateur de dessiner son propre schéma mémoire en composant des primitives graphiques comme des rectangles, des flèches et du texte. L'outil vérifie la cohérence du schéma construit en rapport avec le texte du programme. Par exemple, il s'assure que l'utilisateur ne place pas une flèche, qui symbolise une adresse, dans une case destinée à contenir un réel. Il vérifie également que les objets ont un nombre d'attributs correct. Une fois le schéma construit, l'utilisateur peut le comparer avec la bonne solution fournie par **artEoz**. Cette comparaison peut être automatisée pour donner lieu à une évaluation.

Java ou Python

Le développement initial de notre logiciel a été fait pour le langage java, aussi la version actuelle d'**artEoz** est très complète pour ce langage et propose de nombreuses fonctionnalités de visualisation des mécanismes mis en œuvre (visualisation des champs hérités, mode pas à pas qui permet de dérouler le code instruction par instruction, visualisation de la pile à l'exécution incluant les appels des fonctions : receveur et paramètres, ...)

Nous avons très récemment commencé à intégrer la possibilité de saisir du code python, incluant l'utilisation de modules python externes et nous proposons sur notre site web une version d'**artEoz** adaptée à Python. Cette version[2] ne propose pas encore toutes les fonctionnalités de visualisation des différents mécanismes. Le modèle de données utilisé par notre logiciel est facilement transposable d'un langage à l'autre, tout comme le moteur même d'**artEoz**. Aussi nous attendons rapidement des résultats satisfaisants.

Augmenter les fonctionnalités de l'outil

En dehors de l'amélioration de l'ergonomie de l'interface graphique, les possibilités d'extension des fonctionnalités d'**artEoz** sont multiples. Par exemple, afin de varier les possibilités d'utilisation, nous allons intégrer des scénarios génériques, dans lesquels les différentes valeurs pourront être paramétrées. Un scénario permet d'illustrer un mécanisme particulier ; il peut servir de support à une présentation de ce mécanisme et à la réalisation d'exercices. Cela permet de constituer une banque

2 Version 3.0 alpha – juillet 2013.

d'exercices renouvelables à volonté. Une fois cette base de données constituée, nous pourrons mettre en place un mode d'évaluation sous forme de questionnaire.

Conclusion

Le logiciel **artEoz** est né de nos méthodes pédagogiques, il est la transcription logicielle de notre modélisation des concepts que nous demandons à nos étudiants de comprendre puis de maîtriser. Ce logiciel est encore très jeune, nous n'avons pas le recul nécessaire pour juger de sa pertinence, mais nos premières évaluations de ce logiciel par le public concerné par son utilisation sont vraiment très encourageantes. Nos efforts portent actuellement d'une part sur la définition de scénarios d'utilisation afin d'accompagner les formateurs dans leur rôle et d'autre part sur l'adaptation d'**artEoz** au langage Python, plébiscité pour l'initiation à la programmation dans de nombreux établissements.

Le logiciel **artEoz** est déposé à l'Agence pour la Protection des Programmes sous le numéro **IDDN.FR.001.470024.000.R.P.2012.000.10000**. Il est la propriété de l'Université de Lorraine. Le logiciel n'est pas encore exécutable en ligne, mais il est téléchargeable gratuitement depuis le site http://arteoz.loria.fr pour un usage académique après acceptation des termes de la licence d'utilisation. Il se présente sous la forme d'une archive java exécutable, fonctionnant sur les trois architectures de système : Linux, Windows et MacOs.

Bibliographie

Bataller Mascarell, J. (2011). Visual Help to Learn Programming. *ACM Inroads*, 2(4):42–48.

Ben Ari, M..(2001). *Program Visualization in Theory and Practice*.

Ben Ari, M., Bednarik, R., Ben-Bassat Levy, R., Ebel, G., Moreno, A., Myller, N., et al. (2011). A decade of research and development on program animation: The Jeliot experience. *Journal of Visual Languages & Computing*, 22(5), 375 - 384. Available from http://www.sciencedirect.com/science/article/pii/S1045926X11000310

Cross II, J. H., Hendrix, T. D., & Barowski, L. A. (2011). Combining Dynamic Program Viewing and Testing in Early Computing Courses. In: *Proceedings of the 35th Annual IEEE International Computer Software and Applications Conference*, COMPSAC '11, pp. 184–192. IEEE.

Urquiza Fuentes, J., & Velázquez Iturbide, J. Á. (2013). Towards the Effective Use of Educational Program Animations: the Roles of Student's Engagement and Topic Complexity. *Computers & Education*. Available from http://www.sciencedirect.com/science/article/pii/S0360131513000523

Guo, P. J. (2013). Online Python Tutor: embeddable web-based program visualization for computer science education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education.* (pp. 579–584). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/2445196.244536

Kölling, M., & Barnes, D.J. (2012). *Objects First with Java - A Practical Introduction using BlueJ*. Fifth edition, Prentice Hall / Pearson Education. Available from http://www.bluej.org

Moreno, A., Myller, N., Sutinen, E., & Ben Ari, M. (2004). Visualizing Programs with Jeliot 3. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*. Available from http://cs.joensuu.fi/jeliot

Naps, T., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., et al. (2002). Exploring the Role of Visualization and Engagement in Computer Science Education. In *ACM SIGCSE Bulletin*, Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education (Vol. 35, p. 131 – 152).

Sensalire, M., Ogao, P., & Telea, A. (2009). Evaluation of Software Visualization Tools: Lessons learned. In *Proceedings of the 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, VISSOFT'09, Edmonton, Alberta, Canada (p. 19-26).

Sorva, J. (2012). Visual Program Simulation in Introductory Programming Education. Unpublished doctoral dissertation, Aalto University. Available from http://lib.tkk.fi/Diss/2012/isbn9789526046266/isbn9789526046266.pdf

Sorva, J., & Sirkiä, T. (2010). UUhistle: a software tool for visual program simulation. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (pp. 49–54). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1930464.1930471