# Micro-mondes et concurrence par passage de messages : vers une nouvelle façon d'aborder la programmation ?

Cédric Libert[1], Wim Vanhoof[1]
cedric.libert@unamur.be, wim.vanhoof@unamur.be

[1] Université de Namur

**Résumé.** Nous proposons dans cet article de justifier, au regard de la littérature, la pertinence de la question de recherche suivante: une première approche de la programmation qui se base sur la concurrence par passage de messages en utilisant la méthode pédagogique des micromondes enrichis progressivement permet-elle de diminuer le taux d'échec et d'améliorer les compétences algorithmiques des néo-programmeurs ?

**Mots-clés:** programmation, concurrence, micromondes, passage de messages, didactique

## Introduction

Le premier cours de programmation constitue un enjeu important dans le cursus en sciences informatiques. En effet, cette discipline est au cœur de la plupart des autres cours et permet de développer chez les étudiants des compétences essentielles telles que la résolution de problèmes, l'abstraction et la décomposition (ACM/IEEE-CS Joint Task Force on Computing Curricula, 2013, p.41) .

Toutefois, bien que le taux d'échec dans ce cours ne soit pas globalement alarmant (Watson & Li, 2014), il reste assez élevé et contribuerait, en général, à décourager beaucoup d'étudiants d'entreprendre les études d'informatique (Kelleher & Pausch, 2007). Ce constat mène à des initiatives adaptées aux plus jeunes telles que l'utilisation du langage Scratch (Maloney et al., 2010) avec des enfants (Maloney et al., 2008; Franklin et al., 2013) , le développement de Coder Dojos ou de Devoxx4Kids. Nous choisissons de partir de ce même constat pour définir un nouveau cours d'introduction à la programmation destiné à des étudiants de fin d'études secondaires ou début d'études universitaires qui n'ont pas encore programmé. Pour ce faire, nous souhaitons déterminer un contenu original, porté par une méthode pédagogique déjà éprouvée et qui conviendrait à son enseignement.

La question de recherche que nous formulons est la suivante: une première approche de la programmation qui se base sur 1) la programmation concurrente (plusieurs actions ont lieu simultanément) par passage de messages en utilisant 2) l'approche pédagogique des micromondes (mini-langages avec visualisation graphique et ajout progressif de nouveaux concepts et de nouvelles instructions) permet-elle de diminuer le taux d'échec et d'améliorer les aptitudes algorithmiques des néo-programmeurs ? Cette question prend à rebours l'apprentissage initial de la programmation séquentielle, qui est présent dans la plupart des cours d'introduction.

Cet article constitue une étape préliminaire dans notre raisonnement. Il n'a pas pour but de répondre à la question de recherche, mais d'en justifier la pertinence en se basant sur la littérature. Pour ce faire, nous procédons en deux temps. Nous y abordons, dans un premier temps, la concurrence, qui permet de résoudre assez naturellement certaines classes de problèmes. Nous remarquons ensuite que les langages de programmation couramment utilisés sont peu adaptés à l'enseignement de la programmation en général et à celui de la concurrence en particulier. Cela nous conduit, dans un second temps, à présenter un dispositif pédagogique particulier: les micromondes enrichis progressivement.

## Un nouveau contenu: la programmation concurrente

Ce que l'on appelle “concurrence” est la possibilité pour plusieurs actions d'avoir lieu en même temps (Sadowski et al., 2011). Cette caractéristique est présente intrinsèquement dans le monde, où des évènements simultanés ont lieu en permanence. La programmation concurrente capture cette caractéristique en s'affranchissant de l'obligation de séquentialité et en permettant l'exécution (pseudo-)simultanée de plusieurs instructions.

Dans cette section nous décrivons le concept de concurrence et nous présentons un paradigme qui convient bien à son apprentissage: le passage de messages.

### Un concept: la concurrence

Aborder la concurrence dans un cours d'introduction à la programmation est généralement considéré comme plus complexe que rester dans une démarche purement séquentielle (Sutter, 2005; Brabrand, 2008; Carro et al., 2013) . On peut toutefois admettre que les étudiants expérimentent au quotidien un monde concurrent et que cette façon de programmer, si elle est enseignée correctement, pourrait leur sembler assez naturelle (Van Roy et al., 2003)}. Par ailleurs, elle offre, pour certains problèmes, une meilleure performance et/ou une décomposition plus simple en entités indépendantes (Sutter, 2005) . De plus, l'ACM, dans son rapport de 2013 sur l'organisation du curriculum en sciences informatique (ACM/IEEE-CS Joint Task Force on Computing Curricula, 2013, p.44), souligne qu'il serait intéressant de faire de la concurrence dès le premier cours de programmation.

L'idée d'aborder le concept de concurrence dans un premier cours de programmation n'est pas neuve. En effet, Michael Feldman et Bruce Bachus montrent qu'il est possible d'aborder ce sujet chez les novices (Feldman & Bachus, 1997). Lynn Andrea Stein considère, quant à elle, qu'il faut revoir la façon dont on programme. En se basant toujours sur les modèles de Turing/Von Neuman, on donne encore beaucoup d'importance à la métaphore du calcul (“computation as calculation”), selon laquelle tout programme est une fonction. L'auteur souhaiterait que l'on s'oriente plutôt vers la métaphore de l'interaction (“computation as interaction”) où tout programme est une communauté d'entités indépendantes qui communiquent. Cette seconde métaphore correspond beaucoup mieux au fonctionnement de systèmes de haut niveau tels qu'un système d'exploitation ou l'internet (Stein, 1999) .

Le concept de concurrence est présent dans plusieurs paradigmes. Nous en choisissons un où elle est fondamentale et simple à prendre en main: le passage de messages.

### Un paradigme: le passage de messages

Avant de considérer le passage de messages, attardons-nous sur un paradigme qui se révèle simple pour faire de la concurrence: le *concurrent dataflow* (Van Roy & Haridi, 2003). Ce paradigme permet d'éviter tout non-déterminisme, c'est-à-dire que chaque exécution, pour une entrée donnée, fournit le même résultat. On obtient ce paradigme en ajoutant trois concepts au paradigme fonctionnel pur (Doeraene & Van Roy, 2013) :

- les threads, qui permettent de déterminer des blocs d'instructions à exécuter simultanément;
- les variables logiques, c'est-à-dire qui sont soit “pas encore assignées”, soit “assignées une fois pour toutes”;
- la synchronisation sur les variables, un mécanisme qui, lors de la tentative d'évaluation d'une variable qui n'est pas encore assignée, arrête le thread courant jusqu'à ce que la variable soit assignée, puis reprend son exécution lorsqu'il est possible de l'évaluer.

Si le *concurrent dataflow* semble être un paradigme simple pour enseigner la programmation concurrente, il souffre de deux problèmes. Tout d'abord, il n'est pas supporté nativement par beaucoup de langages modernes. Et ceux qui l'ont implémenté l'ont ajouté ultérieurement au moyen de constructions syntaxiques complexes. Cela peut donc poser problème pour l'enseignement, dans la mesure où une syntaxe claire favorise l'assimilation

en mettant en avant plutôt la réflexion algorithmique que la résolution de problèmes syntaxiques (Mciver, 2000). Ensuite, le problème le plus important est que l'impossibilité de non-déterminisme n'est pas toujours désirée. En effet, le non-déterminisme est nécessaire pour écrire certaines applications concurrentes classiques, telles qu'un client-serveur. En effet, dans ce genre d'application, l'ordre dans lequel les clients contactent le serveur n'est pas déterminé à l'avance (Van Roy & others, 2009) .

Ces deux raisons nous conduisent à passer du concurrent dataflow au passage de messages en permettant, dans certaines situations, une forme de non-déterminisme. Il suffit, pour cela, d'ajouter un mécanisme simple qui crée du non-déterminisme: des canaux de communications (ou "ports") à utiliser pour envoyer et recevoir des messages, et où l'ordre de réception des messages est précisément non-déterministe. On entre dans le paradigme dit de passage de messages. Ce paradigme permet de créer des entités indépendantes et concurrentes, chacune présente dans un thread, qui interagissent en envoyant et en recevant des messages via les ports. Le message reçu est traité et entraine une action en fonction du comportement défini dans l'entité.

Nous proposons ainsi d'aborder la concurrence dès le premier cours de programmation, comme suggéré par plusieurs articles dans la littérature (Feldman & Bachus, 1997; Stein, 1999; Van Roy et al., 2003). Nous pensons que le paradigme du passage de messages se prête bien à cet apprentissage, mais qu'il nécessite d'être soutenu par une méthode pédagogique particulière: les micromondes enrichis progressivement.

## Une méthode: les micromondes enrichis progressivement

Comme le souligne Tony Jenkins (Jenkins, 2002) , la programmation est une matière complexe à enseigner. En effet, cette matière requière le développement de beaucoup d'aptitudes et enseigne des concepts qui se basent les uns sur les autres. De plus, les étudiants perçoivent souvent ce cours comme ennuyeux et difficile à aborder, notamment lorsque les langages enseignés ne sont pas destinés à l'enseignement et ont une syntaxe peu naturelle. L'approche pédagogique par micromondes permet de pallier ces problèmes tout en étant adaptée à l'apprentissage de la concurrence.

### Les micromondes

Conceptuellement, un micromonde est un environnement d'apprentissage construit pour améliorer les mécanismes cognitifs liés à ce que l'on souhaite enseigner (Papert, 1980). Cela favorise chez les étudiants la découverte et l'assimilation de nouveaux concepts.

Dans le cadre d'un cours d'introduction à la programmation, un micromonde est généralement composé de:

- un langage de programmation. Idéalement, il devrait offrir une syntaxe et une sémantique simple;
- une visualisation graphique. Dans notre cas, elle consiste à montrer les entités qui communiquent et agissent en fonction des instructions transmises.

Ces deux éléments sont généralement intégrés dans une interface unifiée qui allie un éditeur de texte et une fenêtre de visualisation. On retrouve cela dans des langages tels que Logo (Feurzeig & Bolt, 1969 ) et ObjectKarel (Satratzemi et al., 2003).

### Intérêt pédagogique

L'intérêt de ce type d'environnement est de favoriser un apprentissage progressif chez l'étudiant, grâce à un contrôle de l'enseignant sur les nouveaux concepts à aborder. Au début, l'étudiant dispose de peu d'instructions pour résoudre les problèmes. Puis, l'enseignant le met face à un problème difficile à résoudre avec seulement ces instructions. Suite à cela, l'enseignant introduit un nouveau concept qui enrichit le micromonde. Cela n'est pas possible dans un langage traditionnel où tout est accessible tout de suite. Stelios Xinogalos (Xinogalos, 2012) et Linda Mciver (Mciver, 2000) relèvent ainsi un certain nombre de problèmes pédagogiques liés à

l'utilisation des langages traditionnels pour l'enseignement de la programmation. Ces problèmes peuvent généralement être résolus par l'utilisation de micromondes.

Le premier problème des langages de programmation courants est qu'ils disposent d'un ensemble large et complexe d'instructions. Le micromonde, quant à lui, constitue un mini-langage comprenant peu d'instructions, ce qui permet aux étudiants de ne pas se perdre dans un grand langage. On y ajoute ensuite progressivement d'autres instructions pour que l'étudiant acquière de nouveaux concepts. Un second problème est que les étudiants se concentrent sur la syntaxe des langages courants plutôt que sur la résolution de problèmes. La syntaxe et la sémantique du langage lié au micromonde devraient être simples et claires. Le troisième problème est l'exécution généralement cachée des programmes courants, où il est difficile pour les étudiants de voir le déroulement des instructions et le résultat de ce qu'ils codent, et donc de comprendre les structures de contrôle et la nature dynamique du programme. La visualisation des agents du micromonde permet, par contre, à l'étudiant de voir immédiatement l'effet des instructions. Finalement, les nombres et les symboles, qui constituent la plupart des problèmes à résoudre avec les langages classiques, sont loin de leur vie de tous les jours, moins amusants et plus difficiles à appréhender. Réaliser des algorithmes sur une communauté d'agents visuels qui exécutent les actions demandées est peut-être, dans un premier temps, une bonne façon d'aborder les problèmes.

L'approche par micromondes permet ainsi d'éviter les difficultés des langages classiques. Les résultats de Stelios Xinogalos vont en ce sens: les étudiants ont une meilleure compréhension des concepts dans un vrai langage après avoir travaillé avec un micromonde (Xinogalos, 2012). De plus, nous pensons qu'une visualisation d'entités qui peuvent agir en même temps permet plus simplement de comprendre le concept fondamental que nous souhaitons enseigner: la concurrence.

## Conclusion

Notre point de départ est un problème régulièrement soulevé dans la littérature: un taux d'échec relativement élevé chez les étudiants qui terminent leur premier cours d'introduction à la programmation (Watson & Li, 2014) et un taux d'inscription aux études de sciences informatiques qui décroît (Kelleher & Pausch, 2007). Certains auteurs mettent au point des approches permettant de diminuer ces problèmes: la programmation par paires (McDowell et al., 2006), l'instruction par les pairs (Porter & Simon, 2013), un plus grand encadrement des travaux pratiques associé à un environnement d'e-learning graphique et à un apprentissage progressif (Boyle et al., 2003), ou encore un changement de langage (Python plutôt que C (Nikula et al., 2011) ou Java (Koulouri et al., 2015) par exemple).

Nous proposons une autre façon de modifier ce cours pour enrayer ce problème, en en déterminant un nouveau contenu et en y adaptant la méthode pédagogique. Le contenu, pour nous, devrait être orienté autour de la concurrence par envoi de messages, qui a suscité l'intérêt de plusieurs auteurs mais dont l'enseignement est très peu étudié dans la littérature. Nous pensons que la méthode pédagogique des micromondes enrichis progressivement convient bien à l'enseignement de ce concept. Nous envisageons donc, pour la suite, de mettre en place et d'évaluer un cours d'introduction à la programmation basé sur la concurrence et enseigné grâce à un micromonde enrichi progressivement de manière à répondre à la question de recherche.

## Références


ACM/IEEE-CS Joint Task Force on Computing Curricula (2013). Computer Science Curricula 2013.

Boyle, T., Bradley, C., Chalk, P., Jones, R., Pickard, P. (2003). Using Blended Learning to Improve Student Success Rates in Learning to Program. *Journal of Educational Media*, Vol(28), 165-178.

Brabrand, C. (2008). Constructive Alignment for Teaching Model-Based Design for Concurrency. In: Jensen, K., van der Aalst, W., Billington, J. (Ed.), *Transactions on Petri Nets and Other Models of Concurrency I*, Springer Berlin Heidelberg.

Carro, M., Herranz, Mari no, J. (2013). A Model-driven Approach to Teaching Concurrency. *Trans. Comput. Educ.*, Vol(13), 5:1-5:19.

Doeraene, S., Van Roy, P. (2013). A new concurrency model for Scala based on a declarative dataflow core.

Feldman, M.B., Bachus, B.D. (1997). Concurrent Programming CAN Be Introduced into the Lower-level Undergraduate Curriculum. *SIGCSE Bull.*, Vol(29), 77-79.

Feurzeig, W., Bolt, B. ( 1969 ). *Programming-Languages as a Conceptual Framework for Teaching Mathematics. Final Report on the First Fifteen Months of the LOGO Project*. Distributed by ERIC Clearinghouse [Washington, D.C.].

Franklin, D., Conrad, P., Boe, B., Nilsen, K., Hill, C., Len, M., Dreschler, G., Aldana, G., Almeida-Tanaka, P., Kiefer, B., Laird, C., Lopez, F., Pham, C., Suarez, J. & Waite, R (2013). Assessment of Computer Science Learning in a Scratch-based Outreach Program. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education, ACM*, 371-376.

Jenkins, T. (2002). On the Difficulty of Learning to Program.

Kelleher, C., Pausch, R. (2007). Using Storytelling to Motivate Programming. *Commun. ACM*, Vol(50), 58-64.

Koulouri, T., Lauria, S., Macredie, R.D. (2015). Teaching Introductory Programming: a Quantitative Evaluation of Different Approaches. *#j-TOCE#*, Vol(14).

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., Rusk, N. (2008). Programming by Choice: Urban Youth Learning Programming with Scratch. *SIGCSE Bull., ACM*, Vol(40), 367-371.

Maloney, J., Resnick, M., Rusk, N., Silverman, B. & Eastmond, E (2010). The Scratch Programming Language and Environment. *Trans. Comput. Educ., ACM*, Vol(10), 16:1-16:15.

McDowell, C., Werner, L., Bullock, H.E., Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, Vol(49), 90-95.

Mciver, L. (2000). The Effect of Programming Language on Error Rates of Novice Programmers.

Nikula, U., Gotel, O., Kasurinen, J. (2011). A Motivation Guided Holistic Rehabilitation of the First Programming Course. *Trans. Comput. Educ.*, Vol(11), 24:1-24:38.

Papert, S. (1980). Computer-based microworlds as incubators for powerful ideas. *The computer in the school: Tutor, tool, tutee*.

Porter, L., Simon, B. (2013). Retaining Nearly One-third More Majors with a Trio of Instructional Best Practices in CS1.

Sadowski, C., Ball, T., Bishop, J., Burckhardt, S., Gopalakrishnan, G., Mayo, J., Musuvathi, M., Qadeer, S., Toub, S. (2011). Practical Parallel and Concurrent Programming.

Satratzemi, M., Xinogalos, S., Dagdilelis, V. (2003). An environment for teaching object-oriented programming: objectKarel.

Stein, L.A. (1999). Challenging the Computational Metaphor: Implications for How We Think. *Cybernetics and Systems*, Vol(30), 1-35.

Sutter, H. (2005). The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. *Dr. Dobb's Journal*, Vol(30).

Van Roy, P., Armstrong, J., Flatt, M., Magnusson, B. (2003). The Role of Language Paradigms in Teaching Programming. *SIGCSE Bull.*, Vol(35), 269-270.

Van Roy, P., Haridi, S. (2003). Teaching Programming Broadly and Deeply: The Kernel Language Approach. In: Cassel, L., Reis, R. (Ed.), *Informatics Curricula and Teaching Methods*, Springer US.

Van Roy, P., others (2009). Programming paradigms for dummies: What every programmer should know. *New computational paradigms for computer music*, Vol(104).

Watson, C., Li, F.W. (2014). Failure Rates in Introductory Programming Revisited.

Xinogalos, S. (2012). An evaluation of knowledge transfer from microworld programming to conventional programming. *Journal of Educational Computing Research*, Vol(47), 251-277.