Proposition de recherche doctorale

Paradigmes pour re?concilier ge?ne?ricite? et performance

Mots clés :

- **Directeur de thèse** : Thierry GERAUD
- **Co-encadrant(s)** :
- **Unité de recherche** : Laboratoire de Recherches et Développement de l'EPITA
- **Ecole doctorale** : École Doctorale Informatique, Télécommunications, Électronique de Paris
- **Domaine scientifique principal**: Divers

Résumé du projet de recherche (Langue 1)

Le but d'une approche logicielle générique est de fournir un très haut niveau d'abstraction, par exemple, en séparant la forme générale d'un algorithme des détails spécifiques de ses implémentations potentielles. L'approche orientée objet traditionnelle pose vite deux inconvénients principaux : l'encombrement du code (prolifération des classes et des méthodes), et une dégradation des performances (dispatch dynamique). Il existe aujourd'hui un certain nombre de paradigmes alternatifs à l'approche orientée objet classique permettant d'offrir un haut niveau de généricité sans pour autant sacrifier les performances ou la maintenabilité du code. Certains, comme la méta-programmation, sont bien connus, d'autres, comme la programmation orientée contexte sont plus récent et moins maîtrisés. Dans ce cadre, le but de cette thèse est double : 1. offrir une analyse comparative des paradigmes de programmation touchant de près ou de loin aux problématiques de généricité et/ou de performance, afin d'avoir une vision plus claire de leurs avantages et inconvénients respectifs, 2. envisager un système mélangeant les paradigmes jugés les plus appropriés, afin d'étudier les modalités de leurs interactions. Afin de garantir la validité du travail sur le plan applicatif, la thèse prendra appui sur une application logicielle concrète et non-jouet : une bibliothèque de traitement d'image écrite en Common Lisp (Ansi, 1994). Le choix de Common Lisp est motivé principalement par le fait qu'aujourd'hui encore, il s'agit probablement du langage le plus « multi-paradigme » existant. Sur le plan de la généricité (haut niveau), Common Lisp offre différentes formes de typage, de scoping et d'évaluation. Il dispose d'une couche orientée objet plus expressive que les approches classiques de type C++/Java (Verna, 2008, 2010, 2012). Le caractère dynamique de la couche objet rend par ailleurs disponible certains paradigmes très récents et mal connus tels la programmation orientée contexte (Costanza and Hirschfeld, 2005) ou le dispatch par filtre / prédicats (Costanza et al., 2008; Ucko, 2001) D'autre part, c'est un langage réflexif (offrant à la fois introspection et intercession) et dont la réflexivité est aussi bien structurelle au niveau applicatif, que comportementale au niveau langage (Maes, 1987; Smith, 1984) : Le MOP, « protocole méta-objet » (Paepcke, 1993; Kiczales et al., 1991), et les reader-macros sont autant d'outils permettant au programmeur de modifier jusqu'à la syntaxe et la sémantique du langage lui-me?me, le rendant particulièrement bien adapté à la conception générique de DSLs (Domain Specific Languages, Verna (2012)). Le système de macros de Lisp reste aujourd'hui encore le système de méta-programmation le plus expressif qui soit (Graham, 1993; Hoyte, 2008). Tout ceci est rendu possible par l'homoi?conicité du langage (McIlroy, 1960; Kay, 1969), une propriété quasi inexistence dans les langages plus récents, mais qui rend la manipulation d'expressions d'ordre supérieur simple, directe et immédiate. Sur le plan des performances (bas niveau), Common Lisp dispose de compilateurs (classiques ou « Just in Time ») natifs performants, offre également un typage statique optionnel pluto?t que d'imposer le tout-dynamique, fourni des alternatives destructives (update-in-place) à toutes les opération fonctionnelles pures etc. Tous ces points (et la liste continue) ont un impact direct et immédiat sur les performances (Verna, 2006, 2009). Au final, Common Lisp est l'un des rares langage de programmation à e?tre à la fois haut et bas niveau. C'est ce qui le rend idéal pour une recherche fondamentale visant à réconcilier performances et généricité / expressivité.

Résumé du projet de recherche (Langue 2)

En dépit de tous les avantages prospectifs que l'utilisation de Common Lisp présente, force est de constater qu'à l'heure actuelle, aucun retour d'expérience concret n'a fait l'objet d'une véritable formalisation académique. Plus précisément, la situation actuelle est la suivante. — Common Lisp est un langage très utilisé dans l'industrie (certes, à moindre échelle que C++ ou Java) mais les industriels ne publient pas (ou très peu) sur le plan académique. Il suffit pour s'en convaincre de regarder le contenu des différentes occurrences d'ECLM (European Common Lisp Meeting, une conférence pluto?t orientée industrie). Celui-ci témoigne de l'activité industrielle basée sur ce langage, mais se borne généralement à des comptes rendus applicatifs. — La famille des langages Lisp fait l'objet de recherches académiques, comme en témoigne l'activité des deux conférences principales, ILC (international-lisp-conference.org) et ELS (european-lisp-symposium.org, dont Didier Verna, co-directeur de la thèse envisagée, est l'un des fondateurs). On constate également l'émergence de nouveaux dialectes de Lisp tels Clojure et Racket. Cependant, lorsque de la recherche académique est produite, elle est rarement basée sur Common Lisp. — Enfin, il existe de la recherche fondamentale dans des domaines où Common Lisp est a priori un excellent candidat pour expérimentation, notamment de par son expressivité et son extensibilité. Citons par exemple le champ de la méta-programmation appliquée aux DSLs (Ghosh, 2010; Fowler, 2010). Pourtant, force est de constater que me?me dans ces domaines, Common Lisp est systématiquement absent (oublié ou ignoré ?) de la littérature, au profit d'autres langages plus récents (ce qui ne signifie pas nécessairement plus modernes) tels Haskell, Scala, Lua, Converge, Nemerle, voire C++ (Czarnecki et al., 2004; Fleutot and Tratt, 2007). Le travail de thèse proposé ici vise à corriger ce déséquilibre et, ce faisant, apporter une contribution majeure et novatrice à nos connaissances en matière de langages de programmation, en fournissant des éléments de réponses à la question générale suivante : qu'est-ce que le mélange de paradigmes apporte aux applications devant réconcilier performance et généricité, et où, dans cette optique se situent les avantages et les limites desdits paradigmes et de leurs interfaces ? Encore aujourd'hui, cette question reste peu étudiée dans la mesure où, comme indiqué précédemment, peu de langages offrent autant de paradigmes simultanément que Common Lisp. Finalement, notons que les résultats de ce travail, dans leur aspect de « défrichage », auront une portée dépassant largement le cadre du travail proposé ici. D'un point de vue applicatif, les répercutions seront non seulement valables dans le monde du traitement d'image, mais au delà, pour toute application liée au calcul scientifique numérique en général, interactif en particulier. D'autre part, l'on s'attend à ce que les aspects paradigmatiques puissent également influencer d'autres langages de programmation. Cette dernière hypothèse est justifiée par la simple observation de l'évolution actuelle des langages en question. Certains langages dits modernes (par exemple Ruby) revendiquent explicitement leur héritage Lispien ; les grands langages industriels tels C++ et Java incorporent aujourd'hui progressivement des paradigmes connus en Lisp depuis sa naissance vers la fin des années 60 (le premier ordre fonctionnel en particulier, avec les fonctions anonymes / lambda-expressions etc.) ; d'autres langages plus récents comme C# s'intéressent maintenant au mélange des formes de typage statique / dynamique, une question à laquelle Common Lisp fournissait une réponse possible avec son système de typage statique optionnel en 1984, lors de sa standardisation.

Informations complémentaires (Langue 1)

Sur le plan applicatif, ce travail peut déboucher sur des collaborations avec d'autres équipes travaillant sur le traitement d'images, à commencer par l'équipe OLENA du LRDE (Laboratoire de Recherche et Développement de l'EPITA). D'autre part, il existe un lien étroit entre cette thèse et une thèse CIFRE sur un autre domaine applicatif : les statistiques. À ce sujet, il est déjà prévu de collaborer avec une équipe du MAP5 / Paris-Descartes (CNRS UMR 8145). Sur le plan de l'expressivité, des collaborations sont prévues avec les auteurs de certaines implémentations de paradigmes dynamiques tels ContextL, la couche orientée contexte de Common Lisp, née au département Informatique de la Vrije University de Bruxelle (VUB). Enfin, pour le bas niveau, le travail sur les performances sera très certainement produit en étroite collaboration avec les implémentations libres existantes de Common Lisp, SBCL en particulier, et dont l'un des principaux acteurs se trouve à la Goldsmiths University de Londres.

Informations complémentaires (Langue 2)

Co-encadrant de thèse : Didier Verna, LRDE-EPITA. Références Abelson, H., Sussman, G. J., and Sussman, J. (1996). Structure and Interpretation of Computer Programs. MIT Press. Becker, R. A., Chambers, J. M., and Wilks, A. R. (1988). The New S Language. Chapman & Hall. Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2014). Julia : A fresh approach to numerical computing. Chavez-Demoulin, V. and McGill, J. (2012). High-frequency financial data modeling using hawkes processes. Journal of Banking and Finance, 36(12) :3415 – 3426. Costanza, P., Herzeel, C., Vallejos, J., and D'Hondt, T. (2008). Filtered dispatch. In Proceedings of the 2008 Symposium on Dynamic Languages, Dynamic Languages Symposium, pages 4 :1–4 :10, New York, NY, USA. ACM. Costanza, P. and Hirschfeld, R. (2005). Language constructs for context-oriented programming : An overview of ContextL. In Proceedings of the 2005 Symposium on Dynamic Languages, Dynamic Languages Symposium, pages 1–10, New York, NY, USA. ACM. Czarnecki, K., O'Donnell, J., Striegnitz, J., and Taha, W. (2004). DSL implementation in MetaOCaml, Template Haskell, and C++. In Lengauer, C., Batory, D., Consel, C., and Odersky, M., editors, Domain-Specific Program Generation, volume 3016 of Lecture Notes in Computer Science, pages 51–72. Springer Berlin / Heidelberg. Fleutot, F. and Tratt, L. (2007). Contrasting compile-time meta-programming in Metalua and Converge. In Workshop on Dynamic Languages and Applications. Fowler, M. (2010). Domain Specific Languages. Addison Wesley. Franchini, S., Vassallo, G., and Sorbello, F. (2010). A brief introduction to Clifford algebra. Technical report, Università degli Studi di Palermo. Ghosh, D. (2010). DSLs in Action. Manning Publications. Graham, P. (1993). On Lisp. Prentice Hall. Halloway, S. and Bedra, A. (2012). Programming Clojure. Pragmatic Bookshelf. Hoyte, D. (2008). Let Over Lambda. HCSW and Hoytech Productions. Ihaka, R. (2010). R : Lessons learned, directions for the future. In Joint Statistical Meetings. Ihaka, R. and Gentleman, R. (1996). R : A language for data analysis and graphics. Journal of Computational and Graphical Statistics, 5(3) :299–314. Ihaka, R. and Temple Lang, D. (2008). Back to the future : Lisp as a base for a statistical computing system. pages 21–33. Kay, A. C. (1969). The Reactive Engine. PhD thesis, University of Hamburg. Kiczales, G. J., des Rivières, J., and Bobrow, D. G. (1991). The Art of the Metaobject Protocol. MIT Press, Cambridge, MA. Maes, P. (1987). Concepts and experiments in computational reflection. In OOPSLA. ACM. McIlroy, M. D. (1960). Macro instruction extensions of compiler languages. Communications of the ACM, 3 :214–220. Norvig, P. (1992). Paradigms of Artificial Intelligence Programming : Case Studies in Common Lisp. Morgan Kaufmann. Paepcke, A. (1993). User-level language crafting – introducing the CLOS metaobject protocol. In Paepcke, A., editor, Object-Oriented Programming : The CLOS Perspective, chapter 3, pages 65–99. MIT Press. Queinnec, C. (2007). Principes d'Implantation de Scheme et Lisp. Paracamplus. Rossini, A. (2009). Common Lisp Statistics : Using history to design better data analysis environments. Smith, B. C. (1984). Reflection and semantics in Lisp. In Symposium on Principles of Programming Languages, pages 23–35. ACM. Sussman, G. J. and Steele, Jr, G. L. (1975). Scheme : An interpreter for extended lambda calculus. Memo 349, MIT Artifical Intelligence Laboratory. Ansi (1994). American National Standard : Programming Language – Common Lisp. ANSI X3.226 :1994 (R1999). Tierney, L. (1990). Lisp-Stat : An Object-Oriented Environment for Statistical Computing and Dynamic Graphics. Wiley. Ucko, A. M. (2001). Predicate dispatching in the common lisp object system. Verna, D. (2006). Beating C in scientific computing applications. In European Lisp Workshop, Nantes, France. Verna, D. (2008). Binary methods programming : the CLOS perspective. Journal of Universal Computer Science, 14. Verna, D. (2009). CLOS efficiency : Instantiation. In International Lisp Conference, pages 76–90, MIT, Cambridge, Massachusetts, USA. Association of Lisp Users. Verna, D. (2010). Revisiting the visitor : the Just Do It pattern. Journal of Universal Computer Science, 16(2) :246–271. Verna, D. (2012). Extensible languages : Blurring the distinction between DSLs and GPLs. In Mernik, M., editor, Formal and Practical Aspects of Domain- Specific Languages : Recent Developments, chapter 1. IGI Global. Verna, D. (2013). The bright side of exceptions. ACCU (Association of C and C++ Users) Conference.