

Pour des architectures logicielles ouvertes et adaptables.

La réflexion : pourquoi et comment ?

P. Cointe et T. Ledoux pour l'équipe OCM (Objets, Composants, Modèles) de l'EMN

Contexte

Aujourd'hui, la complexité et l'ubiquité croissante des systèmes logiciels autant que l'environnement économique compétitif à l'extrême rendent nécessaire le développement de systèmes *ouverts* et *adaptables*. Les logiciels sont de plus en plus complexes, leur réalisation nécessite des investissements de plus en plus importants en temps, et pourtant le délai d'arrivée sur le marché ("*time-to-market*") devient un facteur déterminant de succès.

Aussi, la production de composants logiciels réellement réutilisables est un enjeu tant scientifique qu'industriel. L'ampleur du phénomène Java traduit via l'Internet et le Web l'avènement des technologies des objets et du réseau. Ce phénomène auto-amplifié par le développement du commerce électronique dope le marché des composants (JavaBeans de Sun, ActiveX de Microsoft) ainsi que celui des ateliers de développement et d'intégration (VisualAge d'IBM), des bus logiciels (Corba de l'OMG) et des serveurs d'application (Enterprise JavaBeans de Sun).

Néanmoins, ces environnements ne satisfont que de manière très partielle les besoins des applications. Aussi, il est urgent de spécifier des modèles de composants adaptables, de définir des plates-formes réparties ouvertes pour la production et la diffusion de composants "*on the shelf*".

Dans ce contexte et face à ces enjeux, l'équipe OCM de l'EMN travaille dans deux domaines privilégiés : les langages de programmation et le "*middleware*", et ce, en suivant les méthodologies du génie logiciel par objets. Elle s'intéresse donc tout particulièrement aux architectures logicielles à base d'objets, de composants et de modèles [Coi00] [Ocm00]. Parmi les domaines d'applications figurent bien sûr l'Internet et les télécommunications mobiles.

Proposition

Ce texte constitue une prise de position sur la nécessité de concevoir des noyaux d'architectures, intellectuellement maîtrisable, auto-décrits et extensibles. Nous entendons par *architecture* un terme générique recouvrant aussi bien un langage de programmation, un environnement de développement intégré, un modèle d'objet, un modèle de composants, un modèle de conception, un système réparti, une infrastructure de communication, un serveur d'application, ...

Ce « dogme » s'appuie sur une expérience de plusieurs années développées dans :

- i) l'étude, la définition et l'implémentation de modèles à objets pour les langages à classes, d'acteurs et à prototypes ;
- ii) l'écriture de protocoles de méta-objets (MOP) pour des langages comme CLOS, Self, Smalltalk et Java. Ces protocoles utilisant bien sûr les mécanismes d'héritage et de délégation pour spécialiser ces MOPS ;
- iii) l'étude des architectures à méta-niveaux, et des méthodologies basées sur la "*separation of concerns*" et la programmation par aspects ;
- iv) l'étude des modèles émergents d'objets et de composants répartis.

Historiquement, nous nous sommes d'abord consacrés aux langages à objets dans le but de renforcer leur pouvoir d'expression et de les rendre plus flexibles et plus malléables¹. Aujourd'hui, avec l'émancipation des objets sur le réseau, nous nous intéressons au composants distribués et donc au "middleware", i.e., l'ensemble des services logiciels nécessaires pour permettre et organiser la communication entre applications réparties. Pour comprendre et modéliser ces nouvelles architectures, nous proposons la même démarche que par le passé, à savoir la construction *réflexive* de noyaux expérimentaux. A la manière de B. Smith, nous entendons par réflexif la « capacité illimitée de ce noyau à se représenter et à se manipuler lui même de la même façon qu'il représente et manipule les entités de son domaine d'application ». L'un des enjeux dans la construction de *middleware réflexif* réside dans la réification des mécanismes associés aux différents services (nommage, communication, sécurité, transactions, ...). Ces mécanismes dépassent de loin en complexité ceux proposés par un simple langage à objets (modèles de classes, d'héritage, d'envoi de messages, ...).

Pourquoi ?

Lors du développement d'une architecture logicielle, il existe une contradiction très forte entre le désir de satisfaire les critères de fiabilité, sécurité, portabilité et efficacité, et celui d'introduire des "hooks" (crochets) susceptibles d'adapter ultérieurement cette architecture à des domaines ou des applications non prévisibles à priori. Ces crochets servent finalement à retarder au plus tard les choix les plus pertinents relativement à une stratégie donnée [CoiMal98].

Si l'on prend le cas de Java, dont la conception initiale a clairement bénéficié des expériences accumulées dans les années 75/95 par les principaux concepteurs des langages, on peut effectuer les constats suivants :

- i) le choix a été fait de proposer un modèle de classes simple et de bon goût. En fait, celui de Smalltalk-76. Pourtant pour des raisons de sécurité, ce modèle est non extensible (mot clé *final*) et donc désespérément figé ;
- ii) le besoin d'ajouter au langage des constructions non prévues à priori s'est rapidement fait sentir comme par exemple les classes internes pour réaliser des fermetures, les interfaces pour proposer des réservoirs de constantes. Ces extensions faites de bric et de broc sont ensuite mal intégrées dans le langage, contribuent à accroître la complexité du modèle et rendent difficile sa compréhension ;
- iii) l'utilisation de l'API Reflection², qui introduit des capacités introspectives dans Java, est exploitée de plus en plus régulièrement, en particulier par des outils d'assemblage visuel de composants (de type « BeanBox »), dans le but de découvrir à la volée³ leurs interfaces et leurs propriétés ;
- iv) finalement, on assiste à de nombreuses tentatives visant à « ouvrir » Java en intervenant soit au temps de l'exécution, soit à celui de la compilation, soit finalement à celui du chargement. Ces tentatives correspondent en général au besoin de mieux distinguer et de mieux paramétrer les nouveaux mécanismes introduits pour rendre compte du passage des objets passifs aux objets actifs et finalement aux objets répartis.

¹ « Aujourd'hui, LISP apparaît comme une base de logiciel, une boule d'argile dont le programmeur fait ce qu'il veut ». [Coi87]

² La mal nommée car elle ne supporte pas l'intercession.

³ En reprenant une vieille expression de Jean Louis Durieux, nous pourrions parler de « programmation au lancée » par analogie avec la pêche de même nom!

En conclusion, Java est un bon exemple d'architecture tentant d'assumer ses contradictions, à savoir d'être un langage ouvert et sécuritaire à la fois. Aujourd'hui, un nombre très significatifs de travaux de recherche portent sur la résolution de ces contradictions et donnent sa vitalité à ce langage.

Comment ?

« *L'essence de la programmation réflexive est l'ouverture. Face aux problèmes d'adaptabilité, la programmation réflexive propose qu'une méthodologie générale non seulement transcende toutes les solutions ad hoc, mais préserve l'ouverture des langages et des applications à des modifications ultérieures. Le but ultime de la réflexion appliquée au domaine informatique est donc de définir l'art et la science de la conception et de l'implantation à la fois des langages de programmation et des systèmes informatiques adaptatifs* ». [CoiMal98]

Prenant le relais de la communauté des langages fonctionnels (3Lisp, Brown, Blond), la communauté objet (3KRS, Smalltalk, ObjVlisp, CLOS, ABCL/R, ...) a largement contribué à conceptualiser l'approche réflexive.

Réflexion et protocoles de méta-objets (MOP)

L'une des idées naturelles en programmation par objets est de représenter dans le langage même les différents éléments constitutifs du modèle objet. Par exemple, les classes et leurs membres en Java, les fonctions génériques et les méthodes en CLOS, les classes, leurs méthodes et les messages en Smalltalk. Cette réification conduit à parler de classes de méta-objets (au sens de G. Kiczales), c'est à dire des classes dont les instances représentent ces éléments. Les protocoles associés à ces classes sont qualifiés de protocoles de méta-objets (MOP) et leur spécialisation par héritage ou délégation permet d'adapter le modèle objet à son domaine d'application. Les deux exemples les plus connus de ces protocoles sont le « `lookup0apply` » qui décrit la résolution de l'envoi de messages et le « `allocate0initialize` » qui décrit la construction des objets. Ils sont respectivement à la base de la réflexion de comportement et de la réflexion de structure.

"Separation of concerns" et programmation par aspects

D'un point de vue méthodologique, la "separation of concerns" propose de structurer les applications sur la base du concept d'aspect et favorise ainsi une plus grande réutilisation. Dans le domaine du middleware les différents services pourront être vus comme autant d'aspects : l'aspect sécurité, l'aspect concurrence, l'aspect communication,

La réflexion et ses méta-niveaux liés à l'introduction des méta-objets, introduisent une première séparation entre les opérations relevant des objets de base et celles relevant du programme, de ses types de données et de son exécution. Néanmoins, le même langage est utilisé pour programmer au niveau de base et au niveau méta.

Dans la lignée de la réflexion, la programmation par aspects (*Aspect Oriented Programming*) est apparue en 1996 comme un autre paradigme de programmation, bâti à partir d'une notion de programmes formulés en termes d'aspects spécifiés de manière indépendante dans un langage dédié. Un exécutable est compilé à partir des spécifications des aspects par le moyen d'une technologie de compilation particulière, la « machine à tisser les aspects » (*aspect weaver*).

Evaluation partielle et slicing

D'une certaine manière, la réflexion est une forme d'abstraction allant du plus spécifique au plus générique. Au niveau implémentatoire, il s'agit au contraire – le contexte d'exécution

étant connu – de proposer un processus d’optimisation permettant de supprimer tout code redondant en minimisant les passages entre niveaux. Nous pensons que les techniques de spécialisation de programmes comme l’évaluation partielle et le découpage de programmes (*slicing*) peuvent aider à éliminer cette redondance et donc contribuer à rendre effectivement opérationnelle les architectures réflexives.

Conclusion

Nous cherchons désespérément LA *killer* application !!!

Remerciements

Tous les membres de l’équipe OCM (Jacques Noyé, Mario Sudholt, Rémi Douence, Noury Bouraqadi, Mathias Braux, ...) et ex-membre (Jacques Malenfant) pour les nombreuses discussions relatives à la réflexion comme méthodologie de conception d’architectures ouvertes et adaptables.

Bibliographie

[Coi87] P. Cointe. *Meta-classes are First Class: the ObjVlisp Model*. In Proceedings of OOPSLA'87, Orlando, Florida, December 1987.

[CoiMal98] P. Cointe et Jacques Malenfant. *Aspects avancés de la programmation par objets*, Ecole des jeunes chercheurs en programmation du CNRS, Nantes, 23 mars/3 avril 1998.

[Coi00] P. Cointe. Les langages à objets. *TSI*, 19(1-2-3), 2000.

[Ocm00] Actes de la conférence *OCM 2000* (Objets, Composants, Modèles), « Passé, Présent, Futur », Ecole des Mines de Nantes, 18 mai 2000.

Bibliographie réduite de l'équipe OCM

Réflexion et MOP

F. Rivard

"Smalltalk: a Reflective Language", Reflection'96, San Francisco, 1996.

T. Ledoux, P. Cointe

"Explicit Metaclasses As a Tool for Improving the Design of Class Libraries", ISOTAS'96, JSSST-JAIST, LNCS, vol. 1049, Springer-Verlag, 1996.

P. Cointe

"Reflective Languages and MetaLevel Architectures" , in: "ACM Computing Surveys", vol. 28, no. 4, extended version of [abcc+96a], 1996.

T. Ledoux

"Réflexion dans les systèmes répartis : application à CORBA et Smalltalk" , Université de Nantes, thèse de doctorat, 1998.

M. N. Bouraqadi-Saâdani, T. Ledoux, F. Rivard

"Safe Metaclass Programming", Proceedings of OOPSLA'98, ACM, Vancouver, 1998.

P. Cointe

"Meta-Level Architectures and Reflection (proceedings of the second international conference)", Springer-Verlag, LNCS 1616, 1999.

T. Ledoux
"OpenCorba: a Reflective Open Broker", Reflection'99, LNCS, vol. 1616, Springer Verlag, 1999.

T. Ledoux,, N. Bouraqadi-Saâdani
"Adaptability in Mobile Agent Systems using Reflection" , Workshop on Reflective Middleware, 2000.

R. Douence, M. Südholt
"On the Lightweight and Selective Introduction of Reflective Capabilities in Applications", ECOOP'00 Workshop on ``Reflection and Meta-Level Architectures", 2000.

"Separation of concerns" et programmation par aspects

T. Ledoux
"Adaptabilité dynamique des aspects pour la construction d'applications réparties ouvertes", Colloque International sur les NOuvelles TEchnologies de la REpartition, NOTERE'98, Montréal, 1998.

P. Fradet, M. Südholt
"Towards a generic framework for AOP", Proceedings of the International Workshop on Aspect-Oriented Programming at ECOOP, LNCS, vol. 1543, Springer Verlag, 1998.

P. Fradet, M. Südholt
"An aspect language for robust programming", International Workshop on Aspect-Oriented Programming at ECOOP, 1999.

M. N. Bouraqadi-Saâdani
"Un MOP Smalltalk pour l'étude de la composition et de la compatibilité des métaclasses. Application à la programmation par aspects" , Université de Nantes, thèse de doctorat, 1999.

M. N. Bouraqadi-Saâdani
"Un cadre réflexif pour la programmation par aspects" , Langages et Modèles à Objets (LMO'99), Hermès, 1999.

M. N. Bouraqadi-Saâdani
"Concern Oriented Programming using Reflection", in "Workshop on Advanced Separation of Concerns in Object-Oriented Systems", OOPSLA 2000.

Evaluation partielle et slicing

C. Consel, L. Hornof, J. Lawall, R. Marlet, G. Muller, J. Noyé, S. Thibault, E. N. Volanschi "Partial Evaluation for Software Engineering" , in: "ACM Computing Surveys", vol. 30, no. 3, 1998.

M. Braux
"Speeding up the Meta-level Processing of Java Through Partial Evaluation" , Workshop on Reflective Programming in C++ and Java at OOPSLA, 1998.

M. Braux
"Speeding up the Java Serialization Framework through Partial Evaluation" , ECOOP'99 Workshop on Object Technology for Product Line architectures, 1999.

M. Braux, J. Noyé
"Towards Partial Evaluating Reflection in Java" , ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation, ACM Press, ACM SIGPLAN Notices, 34(11), 2000.

M. Braux, J. Noyé
"Evaluation partielle de la réflexion dans Java", LMO 2000, Hermès, 2000.

M. Braux
"Évaluation partielle de la réflexion dans Java", Université de Nantes, thèse de doctorat, 2000.