

Pour un aspect d'adaptation dans le développement d'applications à base de composants

Thomas Ledoux et Pierre-Charles David

Motivations

Avec la prolifération des plates-formes d'exécution aux caractéristiques techniques parfois très différentes (du téléphone portable aux grilles de calcul) et la démocratisation des réseaux notamment sans fil dans lesquels la disponibilité des ressources (CPU, mémoire, ressources logicielles, ...) peut varier aléatoirement en cours d'exécution, nous sommes plus que jamais confrontés à l'hétérogénéité et la dynamique des environnements dans lesquels nos applications doivent fonctionner.

Devant cet état de fait, nous déclarons que les nouvelles applications réparties doivent pouvoir être adaptées "aisément", voire s'adapter elle-mêmes de façon autonome, aux évolutions de leur contexte d'exécution, dans le but non seulement de continuer à fonctionner correctement (c.à.d garantir les performances ou plus généralement la qualité de service (QoS) attendus), mais aussi de tirer parti de nouvelles possibilités apparues en cours d'exécution. Par le terme "aisément", nous voulons souligner le besoin d'une démarche, d'un cadre méthodologique global (du développement à l'exécution en passant par le déploiement) pour prendre en compte l'adaptation au contexte dans le cycle de vie des applications réparties.

Proposition

Profession de foi

Le besoin de construire des applications qui s'adaptent au contexte n'est pas nouveau. Cependant, cela est généralement fait de façon ad hoc, en tentant d'anticiper au moment du développement les futures conditions d'exécution de l'application, et en intégrant directement dans celle-ci le code nécessaire pour détecter ces évolutions et appliquer les reconfigurations appropriées.

D'un point de vue génie logiciel, cette orientation n'est pas satisfaisante puisque l'observation du contexte et la décision de reconfiguration se trouvent mêlées au code métier, ce qui complexifie le développement, limite les possibilités d'adaptation et réduit surtout la réutilisation de la partie métier. L'adaptation est une "préoccupation transverse", qui *cross-cut* le code métier d'une application, et qui doit être traité séparément du reste. Cette approche inspirée par les concepts de séparation des préoccupations (*separation of concerns*) [1] et programmation par aspects [2] nous invite à repenser le cycle de vie des applications et à considérer l'adaptation aux évolutions du contexte comme un "aspect". Ainsi, dans un premier temps, le développeur d'une application va se concentrer sur sa problématique métier. Puis, une fois connu l'environnement de déploiement et/ou d'exécution, il est possible de définir séparément la logique d'adaptation et de développer l'aspect correspondant. Ce dernier sera finalement tissé (*weavé*) au déploiement et/ou à l'exécution au code métier, produisant ainsi une application auto-adaptable.

Approche choisie

Concrètement, notre objectif est de proposer aux programmeurs d'application un modèle de développement, ainsi qu'un *framework* et des outils associés, permettant de :

– mettre en œuvre cette séparation entre code métier et logique d'adaptation ;

- exprimer le code métier dans un langage, une architecture potentiellement adaptable (c.à.d. flexible et structuré à la fois), et autorisant aussi la reconfiguration dynamique ;
- exprimer la logique d'adaptation dans un langage d'aspect, autorisant un certain nombre de contrôles sur les adaptations réalisables ;
- ré-assembler le code métier et la logique d'adaptation pour obtenir finalement une application auto-adaptable ;
- fournir un service de *context-awareness* permettant de représenter le domaine contextuel à superviser et permettant de détecter les événements qui doivent déclencher l'adaptation.

Solution proposée

Code métier

Pour exprimer le code métier dans une architecture potentiellement adaptable, nous avons réalisé une première expérimentation autour d'un MOP Java [3]. Cependant, le manque de structuration de l'application ainsi réalisée nous a incités à nous tourner vers le paradigme des composants [4]. En effet, les applications développées à base de composants offrent un bon compromis entre flexibilité et structuration : elles supportent certaines reconfigurations (parfois dynamiques) tout en ayant une notion explicite d'architecture (et donc de structure). Plus spécifiquement, nous avons choisi d'utiliser le modèle de composants Fractal [5] comme modèle de base pour nos expérimentations [6].

Logique d'adaptation

La logique d'adaptation représentée par des "politiques d'adaptation" va définir les règles d'adaptation à appliquer aux composants constituant une application. Au cours de son exécution, chaque composant Fractal peut se voir affecter (ou retirer) une ou plusieurs politiques d'adaptations décrivant quand et comment réaliser une adaptation. Le composant Fractal devient alors sensible au contexte et auto-adaptable.

Pour notre langage d'aspect, nous nous sommes inspirés du paradigme ECA (Événement – Condition – Action) issu du domaine des bases de données actives [7] ce qui nous permet de faire un certain nombre de vérifications.

Service de context-awareness

Pour pouvoir adapter une application aux évolutions de son contexte d'exécution, il faut bien évidemment connaître ce contexte. L'objectif de ce service est donc d'observer l'environnement en question et d'être capable d'y détecter les circonstances qui doivent déclencher une adaptation. Ce service sera notamment en charge de l'acquisition des données brutes, la représentation et la structuration des données, le raisonnement sur ces données. Plusieurs domaines contextuels sont actuellement à l'étude (contextes matériel, logiciel). Un protocole push/pull fera l'interface entre le service de *context-awareness* et les politiques d'adaptation.

Conclusion

Pour relever le défi constitué par les « systèmes répartis adaptatifs au contexte », nous proposons une approche basée sur la séparation des préoccupations en considérant *l'adaptation d'une application à un contexte d'exécution spécifique et à ses évolutions* comme un aspect, qui peut et doit être traité séparément du reste de l'application (le code métier). Concrètement, nous avons étendu le modèle de composants Fractal avec un modèle de développement et les outils associés (framework de développement de politiques d'adaptation, service de *context-awareness*) pour créer des composants logiciels adaptatifs au contexte.

Bibliographie

- [1] W. Hürsch and C. V. Lopes – *Separation of concerns*. Technical Report NUCCS-95-03, Northeastern University, Boston, Massachusetts, Feb. 1995.
- [2] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M.Loingtier, and J. Irwin – *Aspect-oriented programming*. In Proceedings of ECOOP, volume 1241 of LNCS, Springer-Verlag, June 1997.
- [3] P.-C. David and T. Ledoux – *An Infrastructure for Adaptable Middleware*. In DOA'02, Springer-Verlag, LNCS 2519, Irvine, California, USA, October 2002.
- [4] C. Szyperski – *Component Software*. Addison-Wesley, 2002. 2nd edition.
- [5] E. Bruneton, T. Coupaye, and J.B. Stefani – *Recursive and Dynamic Software Composition with Sharing*. In Seventh International Workshop on Component-Oriented Programming (WCOP02), ECOOP 2002, Malaga, Spain, June 2002.
- [6] P.-C. David and T. Ledoux – *Towards a Framework for Self-Adaptive Component-Based Applications*. In Proceedings of Distributed Applications and Interoperable Systems 2003, the 4th IFIP WG6.1 International Conference, DAIS 2003, volume 2893 of LNCS, pages 1-14, Springer-Verlag, Paris, November 2003.
- [7] K. R. Dittrich, S. Gatzju, and A. Geppert – *The active database management system manifesto: A rulebase of a ADBMS features*. In Proceedings of the 2nd International Workshop on Rules in Database Systems, volume 985, pages 3–20. Springer-Verlag, 1995.