JCS'2000
Besançon, 21 Juin 2000
Journée des Jeunes Chercheurs en Systèmes

Environnement de programmation actif pour la mobilité*

Christian Khoury, Bertil Folliot
LIP6 - SRC, Université Paris 6
4 Place Jussieu F-75252 Cedex

Résumé

Les applications "émergentes" (systèmes embarqués, mondes virtuels) s'appuient sur des solutions à base de mobilité et de partage d'information. Cet article identifie les besoins minimaux pour une "bonne" programmation de telles applications, dénote les insuffisances des systèmes actuels et décrit l'embryon de notre approche pour la conception d'un environnement de programmation et d'exécution "sur mesure" au dessus d'un système hautement réflexif. L'environnement réalisé permettra d'apporter de nouvelles solutions et de nouvelles abstractions à la programmation des applications mobiles/réparties.

1 Motivations

Les applications "émergentes" telles les systèmes embarqués, les cartes à puces, les réseaux actifs ou les mondes virtuels s'appuient sur des solutions à base de mobilité et de partage d'information et ainsi ont du point de vue programmeur les besoins minimaux suivants :

- **Portabilité** du code sur les architectures actuelles et futures. Cela facilite le portage d'une application existante sur une nouvelle architecture et réduit le "time to market" tout en conservant la même sémantique d'exécution.
- **Adaptabilité** statique (à la définition) et dynamique (à l'exécution) de l'environnement d'exécution en fonction de l'évolution des besoins des applications, du matériel ou des contraintes externes dans un souci **d'efficacité**[1].
- **Interopérabilité** des applications avec la possibilité d'échanger des données et des fonctionnalités (réutilisabilité) même pour des applications écrites dans différents langages.
- **Compacité et simplicité** de modélisation de la solution du problème. Il est difficile qu'un seul langage de programmation, surtout généraliste, puisse à lui seul exprimer simplement et d'une façon compacte les abstractions nécessaires à la résolution d'un problème dans un domaine bien spécifique.

*Ce travail est partiellement financé par le MESR, projet RNRT Phénix 99S0361
[1] dans le sens de QoS

- **Sécurité** de l'application vérifiant les quatres propriétés essentielles : authentification, confidentialité, intégrité (sûreté) et disponibilité.

Nous proposons une approche systématique pour le développement d'applications intégrant ces notions. Cet environnement est basé sur une description de machine virtuelle reconfigurable, adaptée au domaine d'application visé.

2 Les insuffisances des solutions actuelles

Les technologies logicielles de la répartition et de la mobilité sont parvenues récemment à un bon degré de maturité. Néanmoins, ces systèmes sont loin de satisfaire **à la fois** tous les points détaillés en 1 :

- Les systèmes réflexifs [14, 7] et les approches à base de μ-noyaux tels les Exokernels [6] ne visent que les besoins en adaptabilité, plutôt de manière statique que dynamique.
- Les machines virtuelles tel Java [8] visent la mobilité et la portabilité
- Les langages dédiés tels PLAN [10] (réseaux actifs), Telescript [9] (agents mobiles) apportent des solutions au niveau de la compacité et de la simplicité de codage en ciblant les instructions du langage pour un domaine précis
- Les solutions à base de composants (EJB [2], COM/DCOM [1], etc) ainsi que les middleware du style CORBA [11, 5] et DCE [12] visent essentiellement les besoins en interopérabilité

Les travaux actuels proposent, dans chaque domaine, des solutions intéressantes, mais néanmoins incomplètes. Par exemple, pour ajouter la persistance à Java, l'adapter pour des problèmes temps réel ou pour les systèmes embarqués, il faut changer la machine virtuelle. La portabilité du code, ancien et nouveau, n'est plus forcément assurée. En plus, actuellement, cette adaptation est faite au cas par cas et d'une façon statique. Quant à la programmation, elle est limitée au langage Java qui n'est pas nécessairement adapté au problème (i.e, Java Card [3] pour les cartes à Puces).

Finalement, pour résoudre un problème donné en utilisant les technologies actuelles tout en assurant les propriétés énoncées en 1 , il faudrait combiner, si possible, toutes ces technologies. Or, ceci induit une complexité de réalisation importante sans oublier qu'il faut alors déformer la solution en fonction des outils utilisés et pas le contraire. Tout au plus assiste-t-on à des tentatives ad hoc et morcelées d'extension ou d'adaptation de ces plates-formes à chaque domaine rencontré.

3 Objectifs

L'ambition de ce projet est de pallier aux inconvénients des approches “classiques” (morcellement des plates-formes par domaine, incompatibilité des extensions entre elles, difficultés de programmation, complexité des plates-formes existantes). Notre objectif est d'exprimer une solution bénéficiant des propriétés souhaitées dans un “système sur mesure” où l'application aurait le contrôle sur les ressources systèmes et un ou plusieurs langages présentant les abstractions nécessaires pour la modélisation de cette solution sans pour autant avoir à manipuler divers outils incompatibles et complexes.

Nous basons ces travaux sur la plate-forme MVV [4] qui se caractérise par une réflexivité à tous les niveaux (système et langage). Ceci permet une sélectivité à granularité fine des différentes briques formant l'environnement de programmation et d'exécution, i.e la gestion mémoire, l'ordonnancement, le langage de programmation. Nous construirons au dessus un environnement de programmation (langage) et d'exécution (système) actif adapté à la mobilité décrit sous forme d'une MVlet (voir Section 5) .

Dans le paragraphe suivant, nous montrons à travers un exemple simple l'utilité de l'adaptation dans l'expression d'une solution.

4 Exemple des besoins d'adaptation : les réseaux actifs

Soit une application (i.e, mise à jour d'un routeur) qui envoie un paquet *P1* nécessitant un traitement *CODE1* N fois , et d'autres paquets *P2*... *k* necessitant respectivement les traitements *CODE2*... *k* une fois. Ces codes ne se trouvent pas auparavant sur les routeurs. Pour cela, le programmeur décide d'utiliser des réseaux actifs similaires à PLAN [10] et ANTS [13], qu'on appelera par la suite respectivement *RA1* et *RA2*. Les réseaux actifs se proposent de répondre au manque de flexibilité des réseaux traditionnels, en rendant les infrastructures programmables, i.e possibilité d'exécution de programmes applicatifs sur les routeurs.

Les paquets dans RA1 sont formés d'un champ *DATA* et d'un champ *CODE* (appelée dans la littérature approche intégrée). La machine virtuelle de ce réseau actif fonctionne comme suit:

```
Réception Paquet: exécuter CODE sur DATA
```

Les paquets dans RA2 sont formés d'un champ *DATA* et d'un identificateur de code *IDCx* (appelée dans la littérature approche discrète). La machine virtuelle de ce réseau actif fonctionne comme suit:

```
INIT: Envoyer les traitements nécessaires à l'application à tous les sites

Réception Paquet: Appliquer CODEx correspondant à IDCx aux données DATA

FIN : Supprimer les codes
```

Si l'application utilise *RA1*, on aura *N* émissions superflues de *CODE1*. Dans le cas de *RA2*, on perd les ressources utilisées par les codes *CODE2* à *CODEk* exécutés juste une fois. Si l'application réseau actif utilise tout le temps le même code, il serait redondant de l'envoyer à chaque fois avec les données et alors l'approche *RA2* est plus efficace. En revanche, si elle utilise pour chaque paquet un code différent, l'ensemble de tous ces codes peut ne pas tenir en mémoire du routeur et donc l'approche *RA1* est préférable.

Cet exemple montre qu'un seul protocole de réseau actif n'est pas capable de satisfaire aux particularités de toutes les applications. Dans ce cas précis, c'est un compromis ressources locales/ressources réseaux optimum adapté à l'application qu'on cherche à mettre en place. Or une particularité applicative ne pourrait être détectée sans que l'application elle-même (le programmeur) ne déclare ses propres besoins/particularités.

La solution idéale dans ce cas particulier serait de combiner les deux approches de façon à utiliser une solution du type *RA1* pour les codes fréquemment utilisés et une stratégie type *RA2* pour les codes utilisés pour une seule fois sans pour autant interrompre le routeur.

Mais, notre exemple n'est pas réaliste; on ne peut prédire le comportement d'un administrateur pour la mise à jour d'un routeur. Et si l'application avait besoin d'un troisième protocole pour optimiser ses traitements ? Et si l'environnement d'exécution évoluait (i.e, connexion a débit variable) ? etc...

Tout ceci nous montre la nécessité de pouvoir concevoir une application capable de s'adapter (utilisation de plusieurs protocoles) et d'adapter son environnement (lancement de machines virtuelles correspondant à ces nouveaux protocoles) à la volée pour mieux s'exécuter. Pour cela, il faudrait définir un environnement permettant à un programmeur de concevoir des applications actives du type décrit dans notre exemple. C'est à terme ce que permettra notre **environnement d'exécution actif**.

5 Environnement d'exécution actif pour la mobilité

Notre projet porte sur la généralisation de la solution visée dans la section précédente pour arriver à définir une MVlet générique pour le domaine précis de la mobilité.

Une **MVlet** est une description de haut-niveau (dans un langage dédié) des composants d'une machine virtuelle. Chaque spécification est composée de :

- une définition déclarative du modèle d'exécution,
- une implantation de la "colle" nécessaire entre le modèle d'exécution générique de la MVV et celui de la machine virtuelle décrite,
- une implantation de la "colle" nécessaire entre la mémoire objet générique de la MVV et le modèle mémoire de la machine virtuelle décrite,
- une liste des services nécessaires à la mobilité (bibliothèques),
- une implantation de la "colle" de chargement nécessaire pour charger, décoder et commencer l'interprétation d'une application dans le format standard de cette machine virtuelle,
- une implantation de chaque bytecode.

L'environnement réalisé permettra d'apporter de nouvelles abstractions à la programmation des applications mobiles/réparties basées sur l'existence d'une plate-forme hautement dynamique et réflexive et ainsi montrer l'intérêt de la dynamicité/interopérabilité pour les applications "émergentes". Finalement, et peut-être le point le plus important, conduire à banaliser l'expression de nouvelles solutions en présentant aux programmeurs des environnements simples adaptés à leurs besoins.

Bibliographie

1. COM/DCOM. http://www.microsoft.com/com/.
2. Entreprise Java Beans. http://java.sun.com/products/ejb/.
3. Java Card Forum. http://www.javacardforum.org/.
4. Machine Virtuelle Virtuelle. http://www-sor.inria.fr/projects/vvm.
5. Fabio Costa, Gordon Blair, and Geoff Coulson. Experiments with Reflective Middleware. In Springer-Verlag, editor, *Proceedings of the ECOOP Workshop on Reflective Object-Oriented Programming and Systems (ROOPS '98)*, volume 1543 of *Lecture Notes in Computer Science*, pages 390–391, 1998.
6. D. Engler et al. Exokernel: an operating system architecture for application level resource management. In *HOTOS VI*, 1997.
7. Gregor Kiczales et al. *The Art of the Metaobject Protocol*, chapter 5,6. MIT Press, 1991.
8. Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. 1997.
9. General Magic. The Telescript home page. Posted to comp.compilers, 1996. http://www.genmagic.com/Telescript.
10. Jonathan T. Moore, M. Hicks, and Scott Nettles. Chunks in PLAN: Language support for programs as packets. Technical report, University of Pennsylvania, March 1999.
11. Object Management Group, Farmingham, MA. *The Common Object Request Broker: Architecture and Specification*, revision 2.0 edition, July 1995.
12. Open Software Foundation, 11 Cambridge Center, Cambridge, MA. *OSF DCE Application Development Guide*, revision 1.0, update 1.0.2 edition, 1994.
13. Wetherall, Guttag, and Tennenhouse. ANTS: A Tool kit for Building and Dynamically Deploying Network Protocols. In *IEEE OPENARCH'98, San Francisco*, April 1998.
14. Y. Yokote. The apertos reflective operating system: The concept and its implementation. In *OOPSLA*, 1992.