JCS'99
Rennes, 8 juin – 11 juin 1999
Journée des Jeunes Chercheurs en Système

MVV : langage & système, plus qu'un mariage de raison *
VVM: language & system, not just a marriage of convenience

Baillarguet, C.
INRIA, B.P. 105, F-78153 Le Chesnay Cedex (Egalement avec le LIP6)

1 Quels besoins pour quelles applications ?

Les réseaux actifs, les mondes virtuels, les applications réparties avec agents mobiles sont des applications représentatives de l'évolution de l'Internet et de la technologie. Ces applications ont besoin d'un environnement assurant leur mobilité, ainsi que d'un système qui puisse s'adapter ou s'étendre dynamiquement à leurs spécificités. La mobilité requiert un code compact (voyage sain), sûr (provenance de sites étrangers) et portable (hétérogénéité matérielle des plateformes d'accueil). L'adaptation et l'extension de l'environnement d'exécution lui permettent de se modeler en fonction des applications ou du matériel (remplacement ou ajoût de fonctionnalités de granularités diverses). Ces deux propriétés permettent à l'application une homogénéité logicielle ainsi que des conditions d'exécution optimales. La dynamicité assure la réversibilité de modifications mineures ou majeures. Elle autorise également l'évolution du système dans le cas de non atteignabilité (déploiement trop vaste – cartes à puces, ou éloignement physique – satellites). Enfin, un environnement dynamiquement modifiable permet l'homogénéité immédiate des plateformes d'accueil, quelles que soient les conditions matérielles ou logicielles.

Finalement, un système adaptable, extensible, et dynamique, capable d'accueillir des applications mobiles, procurera un environnement à la fois ouvert et spécifiquement modelé pour l'application, que ce soit en termes de performances ou de fonctionnalités.

2 Pourquoi une nouvelle solution ?

L'adaptation, l'extension (voire la dynamicité) des systèmes sont des besoins déjà bien identifiés dans la littérature ([3, 5, 9]), et certaines approches comme les μ-noyaux ou les systèmes réflexifs tentent spécifiquement d'y apporter une solution. Cependant, aucune de ces solutions ne répond à l'ensemble de ces besoins, mais se restreint à un aspect unique, ce qui motive une nouvelle solution.

2.1 Les μ-noyaux

L'approche de base (afin de résoudre le problème de l'adaptation) est de repousser les abstractions du système, sous forme de processus serveurs, dans l'espace utilisateur et de ne garder au niveau du noyau du système que les éléments minimaux (gestion des processus, IPC, etc.). Le problème de cette approche réside essentiellement en une dégradation

*Par contrainte de taille, certaines références bibliographiques sont omises. Une version intégrale de ce papier est disponible sur le site VVM : `http://www-sor.inria.fr/projects/vvm/`

des performances due aux vérifications lors d'exécution de processus utilisateurs comme serveurs de fonctionnalités systèmes. Parallèlement, les modifications sont d'une grosse granularité, puisque pour changer une fonctionnalité, il faut remplacer le serveur entier. Pour y remédier, certains systèmes sont basés sur un mécanisme de bibliothèques (Exokernel [4]) et utilisent une protection logicielle (le μ-noyau vérifie lui même les opérations effectuées sur les ressources). D'autres solutions délèguent à un compilateur certifié la vérification du code produit par l'utilisateur dans des modules systèmes remplaçables (SPIN [2]).

L'inconvénient général de ces approches, outre le problème des performances, est la limitation de la flexibilité, soit à un cadre clairement établi à la conception du système, soit à une déclaration au moment de la compilation de l'application. De plus, chaque μ-noyau ne possède aucune flexibilité propre, et ses caractéritiques sont définitivement établies. D'autre part, ils ne prennent pas en compte les besoins applicatifs actuels, puisqu'ils proposent un environnement natif, hétérogène et non portable, non adéquat à la mobilité des applications. Leur solution est en général spécifique au système sur lequel elle est implantée, et ne serait pas utilisable sur un autre système. Le fait de remonter tout le système dans l'application peut également être discutable dans la mesure où seulement certaines politiques doivent être remplacées, et non le système entier.

2.2 Les systèmes d'exploitation construits autour d'une machine virtuelle

Outre les μ-noyaux qui vérifient leurs scripts d'extensions à l'aide d'interpréteurs (μChoices [10]), certains systèmes intègrent une machine virtuelle (MV) au cœur du système. Fluke [6] utilise la notion de machine virtuelle pour définir l'imbrication de processus, et ainsi gérer les activités systèmes et leur sécurité. J-Kernel [7], utilise la sécurité de Java pour construire un système sûr pour l'accueil simultané d'applications (ou d'applets). JavaOS intègre "nativement" une MV Java, et l'exécute donc plus efficacement (en supprimant le niveau d'indirection dû à l'interprétation). JanOS, qui s'appuie sur le mécanisme d'extensibilité de Fluke, s'intéresse à l'aspect mobilité de Java dans le domaine des réseaux actifs.

L'utilisation de machines virtuelles au sein d'un système est donc fortement liée aux propriétés de celles-ci (sécurité, mobilité), ou au gain de performance envisagé pour les applications Java. Toutefois, en plus du fait que l'extensibilité, l'adaptation et la dynamicité ne sont pas abordés dans la plupart des cas, l'inadéquation pour des langages autres que Java pose un problème fondamental. Il est en effet impossible d'imposer un langage unique et universel pour tous les domaines applicatifs, pour toutes les conditions matérielles et surtout pour tous les paradigmes de programmation.

2.3 Les systèmes réflexifs

Les systèmes réflexifs sont la réponse la plus évidente à la question de l'adaptation. Apertos [12] ou MetaJava modélisent le comportement des objets systèmes par un ensemble d'objets dédiés (les *Meta Objects*). Leur remplacement ou modification assurent l'adaptation du système pour chaque application.

Les altérations du système sont donc en général limitées ou encadrées par l'implémentation. De plus, les langages utilisables sont restreints à celui du système, et la mobilité est très peu abordée. Enfin, malgré l'utilisation de la compilation (qui pénalise la dynamicité), l'approche est très coûteuse pour les performances.

3 La solution MVV

Les systèmes d'exploitation utilisent donc de plus en plus des mécanismes langages ou des machines virtuelles pour s'adapter ou s'étendre parfois dynamiquement. Néanmoins, les solutions sont pour la plupart mono-axiales, et prennent peu en compte la mobilité ou les performances. La Machine Virtuelle Virtuelle (MVV) s'appuie sur une machine virtuelle pour proposer une solution générique et performante à l'ensemble de ces besoins.

3.1 Réconcilier les approches langage et système

Le besoin de mobilité des applications favorise l'utilisation des langages interprétés (code compact, portable, vérifications possibles durant l'exécution pour éviter toute opération illégale), comme le montre l'émergence de Java. Or, Java est un langage généraliste (donc peu performant, même dans le cas de compilation "à la volée"), et doit être spécialisé pour des domaines ou applications exigeants (systèmes embraqués, cartes à puces). Les machines virtuelles ne sont donc pas une réponse en elle-mêmes aux problèmes d'adaptation, d'extension ou de dynamicité des systèmes, et leurs qualités pour la mobilité des applications n'éliminent pas la diversité des langages (domaines, paradigmes, matériels).

Parallèlement, les langages implémentent de plus en plus de mécanismes habituellement réservés au système (processus, persistance). La distinction langage – système devient donc plus difficile [8]. L'idée de la Machine Virtuelle Virtuelle est de centrer le système autour d'une machine virtuelle (sans pour autant imposer un langage de programmation) et de modeler le système résultant pour qu'il soit adaptatif et extensible dynamiquement. L'utilisation des techniques d'optimisation de langages au cœur du système va permettre des performances réalistes (comme expérimenté dans Synthetix [11]).

3.2 Une μ-MV au cœur de la MVV

De façon similaire aux μ-noyaux, la MVV sera composée d'une μ-MV, présentant les caractéristiques minimales d'un environnement d'exécution, et des modules systèmes assurant l'interface avec l'environnement externe (système d'exploitation ou ressources physiques). La μ-MV, basique et générique, comprendra un moteur d'exécution, un modèle mémoire-objet, un ramasse-miettes, un jeu d'instructions et de primitives. L'adaptation au langage d'une application se fera au moyen d'une description de la machine virtuelle concernée.

Une étude de quatre machines virtuelles a montré qu'il existe des mécanismes et un jeu d'instructions minimal communs. A partir d'une μ-MV reposant sur ces éléments, il devrait être possible d'instancier (i.e de modifier dynamiquement la μ-MV pour qu'elle reproduise le comportement préalablement spécifié) des MV applicatives.

3.3 Un système dynamiquement adaptable et extensible

L'adaptation et l'extension des propriétés du système seront réalisées par ajoût ou remplacement de modules dynamiquement liés au moment de l'exécution (aspect proche des bibliothèques dynamiques ou de la réflexion). Un prototype RVM[2]) permet d'étendre dynamiquement le jeu d'instructions ou de primitives en utilisant des fonctions utilisateur écrites dans le langage de l'application, et propose une première technique pour l'extension de la μ-MV. Des travaux sur la description simplifiée de machines virtuelles utilisant cette plateforme ont commencé. Bien que préliminaires, ils montrent la faisabilité de l'approche.

[2]Reflexive Virtual Machine RVM : `http://www-sor.inria.fr/projects/vvm/`

3.4 Et les performances ?

Le double niveau de virtualisation associé à la MVV, dû à l'imbrication de machines virtuelles (une machine virtuelle qui supporte d'autres machines virtuelles), pose un problème réel de performances. Toutefois, la modification dynamique de la μ-MV pour reproduire le comportement d'une MV donnée (insertion dans la μ-MV d'instructions, de primitives ou de mécanismes spécifiques à la MV instanciée) annule cet effet. Les travaux sur le prototype RVM prouvent l'intérêt de cette approche en exhibant un facteur de 2 à 8 entre l'exécution classique et celle avec ajout d'instructions ou de primitives spécifiques à l'application. Ces premiers travaux d'optimisation s'appuieront par la suite sur la génération dynamique de code, approche suivie dans Self et montrant des résultats intéressants.

4 Conclusion

La nécessité d'utiliser une machine virtuelle au cœur du système d'exploitation a déjà été perçue, notament pour profiter de la sécurité ou de la mobilité que peuvent apporter l'interprétation. La MVV cherche à dépasser ce stade et à bâtir autour d'une machine virtuelle un système complètement et dynamiquement adaptatif et extensible, capable de supporter divers langages et de se modeler en fonction des applications ou du matériel.

Aujourd'hui, le cœur de la MVV (la μ-MV) n'est pas complètement dessiné, mais les travaux sur l'extension ou l'adaptation dynamique ainsi que sur les performances y contribuent. L'accent est mis sur une machine capable de créer son jeu d'instructions dynamiquement en fonction de l'application, sur un prototype permettant de mettre en oeuvre ce mécanisme, ainsi que sur la description de machines virtuelles.

Bibliographie

1. C. Baillarguet, *Sémantique des machines virtuelles Smalltalk, Java, Objective Caml et Prolog*, Rapport bibliographique de DEA. `http://www-sor.inria.fr/projects/vvm/`
2. B. Bershad, et al, *Extensibility, safety and performance in the SPIN operating system*, SOSP, 1995.
3. V. Cahill, *Flexibility in object-oriented operating systems : a review*, Technical Report, TCD-CS-96-05, 1996.
4. D. Engler, et al, *Exokernel : an operating system architecture for application level resource management*, HOTOS VI, 1997.
5. D. Engler, et al, *Exterminate all operating system abstractions*, HOTOS V, 1995.
6. B. Ford, et al, *Microkernels meet recursive virtual machines*, OSDI'96, 1996.
7. C. Hawblitzel, et al, *Implementing Multiple Protection Domains in Java*, USENIX 1998.
8. J. Howell, et al, *Hey, You Got Your Language In My Operating System !*, Technical Report PCS-TR98-340, 1998.
9. G. Kiczales, et al, *The need for customizable operating systems*, WWOS'93, IEEE.
10. Y. Li, et al, *Dynamic customization in the μChoices operating system*, 1996.
11. C. Pu, et al, *A Study of Dynamic Optimization Techniques : Lessons and Directions in Kernel Design*, Technical Report OGI-CSE-93-007.
12. Y. Yokote, *The Apertos Reflective Operating System : The Concept and Its Implementation*, OOPSLA 1992.