Titre : Validation formelle de composants système
Projet acceuil : projet EVEREST, INRIA Sophia-Antipolis
Encadrant : Benjamin Grégoire, Chargé de Recherches INRIA

Le développement de l'informatique et des réseaux de télécommunication, en particulier de la téléphonie mobile, offre une très grande flexibilité aux utilisateurs en leur permettant un accès à un grand nombre d'applications et de services. Certains de ces services comme les jeux ne sont pas critiques, d'autres comme le télépaiement ont des demandes très fortes de sécurité. Fournir un support pour ces deux types d'application, sur une unique plate-forme, demande à la fois une grande flexibilité et un haut niveau de sécurité. Ainsi, fournisseurs et utilisateurs expriment des besoins croissants et antagonistes, en flexibilité et sécurité. Nous proposons de développer des formalismes et des outils permettant de garantir un haut niveau de sécurité et de flexibilité pour ces systèmes.

Nous nous concentrerons sur les petits objets portables sécurisés, ou POPSs, fonctionnant sur une plate-forme d'exécution Java et déjà largement utilisés dans des applications bancaires et téléphonique. Il sera donc naturel de concentrer les efforts sur la mise en place de techniques pour le langage Java.

L'approche proposée est basée sur l'utilisation de formalismes de spécification pour la vérification de composants Java, et plus particulièrement sur l'outil de vérification JACK développé au sein du projet EVEREST, que nous utiliserons pour valider des composants système d'une machine virtuelle écrite en JAVA.

JACK (*Java Applet Correctness Kit*) est un environnement de vérification intégré à l'IDE Eclipse permettant de valider des applications Java annotées en JML (*Java Modeling Language*). JACK prend en entrée un programme Java annoté (en JML) et produit à l'aide d'un générateur d'obligations de preuves, des théorèmes pouvant être traités par des outils de démonstration. Les prouveurs proposés par JACK sont principalement automatiques (Atelier B, Simplify, haRVey). Un visualisateur fournit une vision Java des théorèmes à démontrer et les corrèle aux fragments de code annoté dont ils sont issus.

Nous proposons de poursuivre ce développement, afin de compléter cette approche essentiellement automatique, par une autre approche, plus interactive, permettant l'accès à la logique d'ordre supérieur, ceci en conservant un fort niveau d'automatisation. Pour ce faire, nous proposons d'étendre JACK avec un plugin Coq, ce qui fournira un environnement de preuve interactif puissant et d'un niveau de fiabilité très élevé. L'intégration du plugin Coq se fera selon les trois axes suivants :

Extension de JML Le langage JML permet d'exprimer des annotations pour une logique du premier ordre. Cette logique est contrainte par la volonté de JML d'être un langage de spécification et un langage de vérification d'invariants à runtime. Ce double aspect du langage entrane plusieurs lacunes du point de vue de la preuve. Par exemple, les variants de boucle (arguments de terminaison) sont nécessairement des entiers et non une relation d'ordre. De plus, seuls les prédicats *évaluables* peuvent être définis. Nous proposons d'étendre JML, avec des déclarations de prédicats externes. Ceci, permettra, au prix d'une très faible modification de JML, d'accéder à la puissance logique et surtout au pouvoir expressif de la logique d'ordre supérieur. Ces ajouts sont motivés par la nécessité de valider les fonctionnalités des composants système Java comme décrits ci-dessous.

Automatisation des preuves Nous distinguons les propriétés à démontrer avec un outil comme JACK en deux catégories, la première est liée aux annotations faites par le développeur, à ce titre elles sont arbitrairement complexes et potentiellement non automatisables. La seconde correspond aux propriétés de correction *usuelles* générées par JACK, comme la gestion de la mémoire ou les levées d'exceptions. Nous pensons qu'il est impossible d'utiliser, de manière intensive, un prouveur interactif si celui-ci n'a pas une automatisation des preuves appartenant à cette deuxième catégorie proche de cent pour cent.

Cette automatisation des preuves est l'un des points clés de ce doctorat.

Applications à la vérification de composants Nous pensons qu'il est primordial de lier le développement de JACK à des preuves de composants réalistes et de taille conséquente. Ceci permet de tester et d'évaluer l'outil JACK. Nous proposons ici de développer des mécanismes de vérification des fonctionnalités et de l'innocuité des composants système.

Il existe une grande variété de composants système susceptibles d'être étudiés : à titre d'exemple, citons la pile IP (les protocoles de communication sont des composants système qu'il est intéressant de pouvoir charger dynamiquement), les ordonnanceurs, la gestion de la mémoire, ou encore un vérificateur de bytecode Java.