GECCOO

Génération de code certifié pour des applications orientées objet

Spécification, raffinement, preuve et détection d'erreurs

Rapport final
http://geccoo.lri.fr

Décembre 2006-16 janvier 2007

Résumé

Ce document est le rapport final de l'ACI sécurité GECCOO. Il met en évidence les activités de collaboration ayant pris place au sein du projet, en particulier dans le développement de synergies entre les outils et l'étude des mêmes applications suivant les différentes approches. Il décrit les principales avancées et conclut en indiquant les perspectives.

Table des matières

1 Introduction

Le projet GECCOO a officiellement débuté en juillet 2003 pour une durée de 3 ans. À la demande des partenaires, une prolongation a été obtenue jusqu'en janvier 2007.

1.1 Problématique

L'objectif du projet est de proposer des méthodes et des outils pour le développement de programmes orientés objets offrant des garanties fortes de sécurité. Le projet s'intéresse plus spécifiquement à des programmes embarqués sur des cartes à puce ou dans des terminaux qui sont décrits dans des sous-ensembles de Java (par exemple JavaCard).

Notre approche s'appuie sur la spécification des programmes Java à l'aide de formules exprimées dans le langage JML et la possibilité de construire des outils de vérification qui transforment un programme annoté en un ensemble d'obligations de preuve qu'il faut ensuite démontrer.

Le projet s'intéresse à plusieurs problèmes dans cette approche. Tout d'abord, les contraintes de correction et de sécurité des applications doivent être exprimées dès la phase de conception du système. Il convient de les spécifier dans un langage de haut niveau, le système étant ensuite raffiné jusqu'à des programmes exécutables et efficaces. Les étapes de raffinement peuvent nécessiter la résolution d'obligations de preuves. Les obligations de preuves engendrées pour la correction de programmes objets nécessitent de raisonner sur les états de la mémoire, ces preuves sont laborieuses mais assez spécifiques, il est nécessaire de développer des procédures automatiques de preuves adaptées à ces cas.

Enfin il est rare qu'une spécification et un programme soient corrects du premier coup. Le système doit donc détecter le plus tôt possible les erreurs et permettre d'utiliser les obligations de preuve pour engendrer des tests ou produire un code défensif dans le cas où certaines propriétés ne peuvent être garanties de manière statique.

Les méthodes développées dans ce projet ont un champ large d'applications dans les environnements destinés à produire du code objet garanti correct par rapport à des spécifications formelles.

Le projet met en relation des équipes dont l'expertise est diversifiée (preuves automatiques, générateur d'obligations de preuves, modèles objets, raffinement, sécurité des applications JavaCard, génération de tests).

1.2 Participants

1.2.1 Équipe TFC, LIFC, Besançon

Une partie de l'équipe TFC participe au projet INRIA CASSIS commun avec le LORIA à Nancy.

Bellegarde Françoise	Professeur	départ retraite 01/06
Bouquet Fabrice	MCF	HDR en juin 2005
Dadeau Frédéric	Doctorant	Allocation ministère-ACI 10/03-09/06
Giorgetti Alain	MCF	
Groslambert Julien	Doctorant	Allocation ministère 10/04-09/07
Julliand Jacques	Professeur	
Kouchnarenko Olga	Professeur	
Legeard Bruno	Professeur	départ 02/06
Masson Pierre-Alain	MCF	

Sujets de thèse Dadeau Frédéric: *Évaluation symbolique à contraintes pour la validation, application à Java/JML*, soutenance 19/07/06.
Groslambert Julien: *Vérification de propriétés temporelles pour les langages de spécification orientés objets*

1.2.2 Projet CASSIS, LORIA, Nancy

Le projet CASSIS est commun avec le Laboratoire d'informatique de Franche-Comté.

Silvio Ranise	CR INRIA	
Christophe Ringeissen	CR INRIA	
Calogero Zarba	Post-doc	INRIA-ACI 04/04-03/05

1.2.3 Projet Everest, INRIA Sophia-Antipolis

Le projet Everest est issu du projet Lemme qui apparaît dans la proposition GECCOO.

Gilles Barthe	DR INRIA	
Julien Charles	Doctorant	Allocation ministère 10/05-09/08
Benjamin Grégoire	CR INRIA	
Marieke Huisman	CR INRIA	
Mariela Pavlova	Doctorante	bourse INRIA 12/03-10/06

Stagiaires : Alejandro Tamalet (INRIA 'internships')

Sujets de thèses: Julien Charles: *Validations formelles des composants systèmes*
Mariela Pavlova: *Bytecode Verification and its Applications*, soutenance19/1/07.

1.2.4 Projet ProVal, INRIA Futurs & LRI, Orsay

Le projet ProVal est issu du projet LogiCal qui apparaît dans la proposition GECCOO.

Sylvain Conchon	MCF	
Evelyne Contejean	CR CNRS	
Jean-Christophe Filliâtre	CR CNRS	
Claude Marché	DR INRIA	
Aurélien Oudot	Ing. associé	INRIA depuis 09/06
Christine Paulin	Professeur	Délégation+Détachement INRIA depuis 09/05
Nicolas Rousset	Doctorant	CIFRE Gemalto 05/05-05/08

Stagiaires : Nicolas Ayache (Master Recherche 04-09 2005) Vikrant Chaudhary (IIT Dehli, 05-07 2004) Julien Roussel (Epita 3A, 01-06 2005)

Sujets de thèses: Nicolas Rousset: *Conception et validation d'applications embarquées sécurisées.*

1.2.5 Équipe VASCO, LSR, Grenoble

Amal Haddad	Doctorante	bourse libanaise 10/05-09/08
Didier Bert	CR CNRS	
Sylvain Boulmé	MCF	
Marie-Laure Potet	Professeur	
Nicolas Stouls	Doctorant	BDI cofinancée STMicroelectronics 12/03-11/06

Stagiaires. Evelyne Altariba (Ensimag 2A,07-09 2004), Mohamed Hounayda (Miage 2A, 05-06 2004), Xavier Morselli (Master 1, 02-06 2004).

Sujets de thèses Amal Haddad: *Modélisation et vérification de politiques de sécurité: un cadre formel pour les applications embarquées.*
Nicolas Stouls:*Outils formels pour la spécification et le développement de systèmes*, soutenance prévue en juin 2007.

1.2.6 Réunions

Des réunions régulières de travail ont eu lieu environ 3 fois par an qui ont rassemblé l'ensemble des partenaires. Le projet a également suscité des collaborations bi-latérales plus spécialisées :

- Cassis et ProVal autour des outils de démonstration automatique
- Everest et ProVal autour du développement des outils Jack et Krakatoa.
- Everest et TFC autour des propriétés temporelles de programmes JAVA.
- ProVal et Vasco autour de la spécification Demoney et de la notion d'invariant

Le projet a aussi donné lieu à la mobilité de jeunes chercheurs :

– F. Dadeau (TFC) effectue un post-doc en 2006-2007 dans le projet Vasco;
– J-F Couchot (Cassis,TFC) effectue un post-doc en 2006-2007 dans le projet ProVal;
– N. Stouls (Vasco) effectue un ATER en 2006-2007 dans le projet ProVal.

2 Résumé des principales avancées et des résultats obtenus

2.1 Spécifications de haut niveau, raffinement, composition

Un des objectifs du projet était de proposer des langages de spécification de haut niveau pour exprimer des propriétés de sécurité et faciliter ainsi la construction de systèmes garantis corrects.

Nous avons exploré plusieurs approches s'appuyant sur le langage B et sur les programmes JAVA annotés en JML. Nous avons en particulier développé des méthodes et outils pour traduire des propriétés de sécurité exprimées sous forme d'automates, de formules de logique temporelles, ou de politique de sécurité vers des pré et post-conditions portant sur les opérations du système qui peuvent ensuite être vérifiées. Nous avons également montré comment un développement par raffinement permettait de préserver les propriétés de sécurité. Ces travaux sont disponibles à travers les outils GÉNÉSYST, JAG et MECA.

Nous avons également proposé des méthodes de spécification abstraite qui étendent le langage JML et sont adaptées à la vérification statique par preuve, ces travaux sont en cours dans les environnements Jack et Krakatoa d'analyse de programmes JAVA/JML.
Voir sections 3.3 et 3.4.

2.2 Génération et résolution d'obligations de preuves

Durant le projet, plusieurs avancées ont permis l'amélioration de l'automatisation de la résolution des obligations de preuve des programmes JAVA annotés en JML. Les modèles mémoire utilisés ont été affinés pour prendre en compte des propriétés de séparation (allégeant ainsi le travail de preuve). Un modèle mémoire spécifique a été développé pour prendre en compte les spécificités JAVACARD (mémoire persistante et volatile, transactions) permettant ainsi de raisonner sur le comportement en cas d'arrachage de la carte.

Sur le plan de la démonstration automatique, un effort important a été réalisé afin de pouvoir utiliser des outils différents selon un modèle uniforme et de pouvoir ainsi tester leur comportement. Ainsi, un démonstrateur automatique tel que YICES développé récemment et très performant dans les compétitions SMT n'apparaît pas avoir de meilleurs résultats sur les obligations de preuve de programmes que l'ancien démonstrateur SIMPLIFY.

De nombreux résultats théoriques ont été obtenus quant à la possibilité de combiner des théories utiles pour l'analyse de programmes (types énumérés, tableaux indexés, structures chaînées). Des démonstrateurs particuliers (haRVey, RV-SAT, ERGO) ont été développés dans le projet permettant d'expérimenter de nouvelles architectures. Nous avons proposé des traces de preuve pour les outils automatiques qui assurent leur intégration dans des assistants de preuve.
Voir section 3.5.

2.3 Détection d'erreurs

Le projet a permis de transférer des méthodes et outils développés dans le cadre du système B vers les environnements JAVA/JML. Cela a donné naissance à de nouvelles classes d'outils qui n'existaient pas dans le monde JML. On peut ainsi animer les spécifications, en vérifier la cohérence ou bien engendrer automatiquement des jeux de tests obéissant à des critères de couverture.

Ces travaux sont intégrés à l'outil JML-Testing-Tools et utilisent l'outil JML2B, tous deux développés dans le cadre de GECCOO.

Voir section 3.6.

2.4 Chaîne d'outils

Le projet a permis le développement d'outils complémentaires pour l'analyse de propriétés de sécurité de programmes. Des efforts ont été réalisés pour assurer l'interopérabilité des systèmes.
Voir section 3.1.

Le projet a également choisi l'application `Demoney` comme benchmark pour la construction de la spécification et la preuve de propriétés de sécurité d'une application Java.
Voir section 3.2.

3 Présentation des avancées du projet

3.1 Développement d'outils

Les participants du projet développent depuis plusieurs années des outils de modélisation et de preuve. Il s'agit de :

- Jack et Krakatoa, deux générateurs d'obligations de preuve pour des programmes Java spécifiés en Jml.
- HaRVey, un outil pour décider de la validité de formules du premier ordre modulo certaines théories equationnelles utiles à la preuve de programmes.
- BZ-Testing-tools, un animateur et générateur de tests pour des spécifications écrites en B ou en Z.
- GénéSyst, une représentation de systèmes B événementiels par un système de transitions. Basé sur la BoB (Boîte à outils B) qui permet de manipuler les substitutions et d'exécuter des calculs de plus faible précondition.

Le projet regroupe des équipes spécialistes de la méthode B, méthode qui a fait ses preuves en milieu industriel dans le domaine de la spécification et du développement de systèmes sûrs. L'adaptation des techniques et outils spécifiques à la méthode B dans le cadre du développement de programmes objets a été fructueuse.

Un des objectifs du projet était de développer des synergies entre ces outils pour les adapter à la spécification de propriétés de sécurité, la preuve et le test de programmes Java.

3.1.1 Génération de spécifications

Une difficulté liée au traitement de la sécurité des programmes est de traduire des exigences de sécurité en contraintes à vérifier sur le code exécuté. Durant le projet GECCOO, nous avons proposé des solutions concernant des propriétés de sécurité exprimées dans une logique temporelle (outil JAG) et la prise en compte de politiques de contrôle d'accès (outil Meca).

JAG L'outil JAG (JML Annotation Generator) développé par A. Giorgetti et J. Groslambert à Besançon (en interaction avec le projet Everest à Sophia-Antipolis) permet de vérifier des propriétés temporelles sur des classes Java. JAG est composé d'un traducteur de formules exprimées dans une logique temporelle dédiée à Java, introduite dans [D.63] vers des annotations Jml qui assurent que la propriété temporelle est satisfaite. Ces annotations peuvent être vérifiées en utilisant les outils Jack ou Krakatoa.

Le langage de la logique temporelle inspiré des « Dwyers Specification Patterns » traite des terminaisons exceptionnelles et peut exprimer des propriétés de « safety » et de « liveness ». La traduction est décrite en détail dans [D.63] pour la « safety » et dans [E.2] pour la partie « liveness ».

JAG est en cours d'extension à l'ensemble des propriétés temporelles exprimées par des automates de Büchi.

Meca L'outil MECA [D.43] est développé au LSR à Grenoble. Il permet d'engendrer les comportements attendus de fonctions à partir de la description de différentes formes de politiques de contrôle d'accès (politique discrétionnaire, politique obligatoire multi-niveaux, politique basée rôle).

3.1.2 Générateurs d'obligations de preuve

Jack et Krakatoa Les outils Jack et Krakatoa partagent le même objectif de prouver des programmes JAVA annotés par des spécifications JML.

Jack, conçu initialement par Gemplus, est maintenant développé par le projet Everest à l'INRIA Sophia-Antipolis. Jack est né dans un contexte industriel, la motivation initiale était de fournir au développeur un environnement convivial qui intègre la vérification formelle dans le développement d'applets JAVACARD.

Krakatoa est développé par le projet ProVal à Orsay et dispose depuis ses premières versions d'une diffusion libre.

Au début du projet GECCOO, Krakatoa puis Jack ont intégré la génération d'obligations de preuve pour le prouveur automatique SIMPLIFY [D.49], avec des résultats très encourageants. Une sortie pour le prouveur COQ a été développée dans Jack ainsi qu'un ensemble de tactiques spécialisées. L'appel au prouveur interactif de COQ a également été intégré à l'environnement Eclipse de Jack.

Les travaux autour de Krakatoa ont porté en particulier sur la possibilité d'utiliser un ensemble large de démonstrateurs automatiques. Krakatoa tire bénéfice de l'utilisation du langage intermédiaire Why qui permet de factoriser la traduction du modèle et des obligations de preuves vers différents outils de preuve. L'expérimentation de Krakatoa sur l'étude de cas Demoney [E.5] a amené à des modifications de la partie génération d'obligations de preuve (en pratique dans l'outil Why qui sert de base à Krakatoa) ceci afin de faire face à des problèmes d'efficacité.

Les outils Jack et Krakatoa servent également de plate-forme d'expérimentation pour les travaux menés dans le cadre du projet GECCOO en particulier pour le traitement de propriétés de sécurité ou les techniques de démonstration automatique. Ces points seront développés dans les sections correspondantes.

Au-delà de GECCOO Le développement des plate-formes Jack et Krakatoa va au-delà des objectifs initiaux du projet GECCOO.

Les travaux sur Jack sont utilisés dans le cadre du projet européen MOBIUS, en particulier pour le développement de la nouvelle version de ESC-Java. Jack a été étendu pour effectuer des preuves de programmes écrits en byte-code JAVA [D.20, B.2]. L'exemple le plus significatif a été l'utilisation de cet outil pour diminuer la taille du code natif engendré à partir du byte-code [D.28] en justifiant la suppression de tests dynamiques par des techniques de preuve statique.

L'expérience Krakatoa a incité au développement de l'outil CADUCEUS de preuve de programmes C [D.33]. Les travaux en cours portent sur la définition d'un langage commun pour C et JAVA permettant le partage des techniques d'analyse.

Positionnement Il existe plusieurs outils développés ayant des objectifs analogues à ceux de Jack et Krakatoa traitant des langages légèrement différents. On peut citer KeY (`http://www.key-project.org/`), Spec# (`http://research.microsoft.com/specsharp/`), ESC/Java (`http://secure.ucd.ie/products/opensource/ESCJava2/`), Jive (`http://softech.informatik.uni-kl.de/twiki/bin/view/Homepage/Jive`) ...

Ces outils reposent sur des modèles et des architectures variés, leurs objectifs en terme de propriétés à vérifier et d'automatisation diffèrent permettant ainsi un large spectre d'expérimentations. Néanmoins, beaucoup de préoccupations se rejoignent, par exemple concernant le pouvoir d'expressivité des langages de spécifications ou la gestion des invariants.

Des rencontres telles que le workshop « Challenges in Java Program Verification » `http://www.cs.ru.nl/~woj/esfws06/` permettent l'échange et la confrontation des approches.

3.1.3 Démonstrateurs automatiques

haRVey L'utilisation de méthodes de preuve automatique est essentielle dans l'architecture des outils de spécification et preuve de programmes. Le démonstrateur Simplify développé par Compaq dans le cadre de l'outil d'analyse de programmes ESC/Java donne de très bons résultats, mais on peut espérer faire mieux en terme d'efficacité et de traçabilité des preuves.

haRVey [D.31] est développé dans le projet Cassis à Nancy en collaboration avec David Déharbe (DIMAp/UFRN, Natal, Brazil). L'outil prend comme entrée une théorie T et des formules ϕ_i et vérifie que les formules ϕ_i sont vérifiées dans la théorie T.

Durant le projet, deux versions de haRVey ont été développées : haRVey-FOL et haRVey-SAT. haRVey-FOL intègre un solveur booléen, un démonstrateur équationnel et une procédure de décision pour l'arithmétique linéaire en suivant l'approche de [D.46]. haRVey-SAT intègre un SAT-solver et une combinaison (à la Nelson-Oppen) de procédures de décision pour la théorie des symboles de fonction non-interprétés et l'arithmétique linéaire avec de plus des techniques pour instancier des quantificateurs et des lambda-expressions. De plus haRVey-SAT peut produire des traces de preuves pour l'assistant Isabelle/HOL. haRVey-FOL offre un bon degré d'automatisation sur une large classe de théories alors que haRVey-SAT est en général plus rapide sur des théories simples et produit des preuves sûres qui peuvent être intégrées dans un environnement interactif. L'objectif à court terme est de combiner ces deux systèmes.

Durant le projet, l'outil Krakatoa a été adapté pour fournir des obligations de preuve pour les deux versions de haRVey.

Ergo Ergo est un outil de démonstration automatique dédié à la résolution d'obligations de preuve. Il est développé par Sylvain Conchon et Evelyne Contejean dans le projet ProVal à Orsay. Il utilise un algorithme de « Congruence Closure » paramétré par une théorie (actuellement l'arithmétique linéaire). Son originalité est de traiter directement une logique multi-sortée polymorphe qui correspond au langage de l'outil Why. C'est également un code très court (3000 lignes de Ocaml) et proche de la description logique de la procédure de décision ce qui est un élément important pour la certification et l'intégration à des démonstrateurs tels que Coq.

Ergo est distribué sous licence GPL `http://ergo.lri.fr`

3.1.4 Analyse de spécifications

JML-Testing-Tools L'outil JML-Testing-Tools est réalisé à Besançon. Il utilise des techniques développées dans le cadre de BZ-Testing-tools au cas des spécifications Jml.

Cet outil utilise un système de résolution de contraintes ensemblistes et permet d'animer les spécifications afin de valider le modèle. Il s'appuie sur un format intermédiaire basé sur une notation ensembliste inspirée de B pour décrire les comportements des spécifications JML et ainsi permettre leur animation symbolique à contraintes, tel que présenté dans [D.13, D.14].

Génésyst Génésyst est un outil développé au-dessus de la boite à outils BOB. Il permet de représenter un système B événementiel par un système de transitions étiquetées. La description peut correspondre à une spécification ou un raffinement. L'utilisateur caractérise dans la description B, l'ensemble des états qu'il souhaite voir apparaître. Le système GénéSyst produit alors des obligations de preuve permettant de calculer les conditions de franchissement des transitions en terme de déclenchabilité et d'atteignabilité.

Le fonctionnement de GÉNÉSYST est décrit dans [D.54, D.5]. Il est disponible en téléchargement[1] sous la licence CeCILL.

Des travaux ont été menés pour pouvoir utiliser d'autres démonstrateurs automatiques que celui intégré dans l'atelier B. L'adaptation de Barvey (interface développée au LIFC pour le démonstrateur haRVey du Loria) a du être reportée ultérieurement. C'est finalement le démonstrateur automatique de l'outil B4FREE qui a été adapté à GénéSyst.

Cet outil a été expérimenté sur l'application Demoney.

JML2B: Obligations de preuves pour la vérification de modèles Jml Dans le cadre de la vérification de programmes basée sur la modélisation JML, il est important de s'assurer que le modèle JML de l'application ne comporte pas d'incohérence. Bien qu'il existe de nombreux outils de vérification de la cohérence du code Java vis-à-vis de la spécification JML, il n'existait pas d'outil permettant de s'assurer qu'il n'y a pas d'incohérence dans le modèle JML lui-même. Nous avons proposé une technique de vérification basée sur une traduction de la spécification JML vers le langage des machines abstraites B. La cohérence de la machine B générée, vérifiée avec l'Atelier B, implique la cohérence du modèle JML original. Ce travail a été publié à la conférence ZB'2005 [D.11] et a donné lieu à l'implantation du prototype JML2B [D.12]. Celui-ci a permis de vérifier la cohérence du modèle JML de Demoney utilisé pour la génération de tests [D.10].

3.2 Étude de cas : Demoney

Demoney est une applet JAVACARD développée par la société Trusted Logic dans le cadre du projet européen SecSafe. Cette applet implante un porte-monnaie électronique, elle a été développée à des fins d'expérimentation d'outils de recherche. C'est donc une version simplifiée par rapport aux applets industrielles mais néanmoins représentative des principales difficultés.

Nous avons choisi dans le projet d'étudier plus précisément cette application. Nous avons analysé le document de spécification. Nous avons ensuite réalisé plusieurs modélisations formelles de cette application.

- À Grenoble, un travail a été réalisé autour de la spécification de la spécification de Demoney qui a été modélisée en B en se concentrant sur les messages d'erreur et les différents états des transactions et des canaux de communication. Les sources de cette modélisation sont disponibles `http://www-lsr.imag.fr/users/Nicolas.Stouls/`. Cette modélisation a servi de test pour l'outil GENESYST [D.5].
- À Orsay, une spécification JML du code Java a été réalisée [E.5] et prouvée correcte à l'aide de l'outil Krakatoa.

[1] http://www-lsr.imag.fr/Les.Personnes/Nicolas.Stouls/

– À Besançon, l'application Demoney a été spécifiée par raffinement en JML. Il s'agit d'une démarche de « spécification pure », réalisée indépendamment de toute idée d'implantation et à partir de la seule spécification publique de Demoney. Les objectifs de l'expérience étaient les suivants :
  – évaluer la pertinence de la démarche de spécification par raffinement en JML,
  – disposer d'une spécification très détaillée de Demoney pouvant servir à l'expérimentation des différents outils réalisés dans le cadre du projet,
  – comparer les tailles respectives d'un modèle de spécification et d'une implémentation de Demoney.

  Le modèle le plus abstrait, point de départ de la démarche de spécification par raffinement, a été simplement constitué par la liste des commandes de Demoney. Puis les détails de la spécification publique ont été introduits un à un, du plus intuitif au plus technique, chaque détail introduit donnant naissance à un nouveau modèle JML raffinant le précédent.

  Au final, 13 niveaux de raffinement ont été développés, jusqu'à la spécification de la communication par APDU mise en œuvre dans Demoney. La spécification a pu être réalisée en n'utilisant que des fonctionnalités courantes de JML, ce qui valide le fait qu'une telle démarche est aisément adaptable à grande échelle. L'expérience a également montré qu'une démarche de spécification incrémentale est valide, dans le sens où il n'a pas été nécessaire de modifier *a posteriori* les niveaux abstraits au fur et à mesure de l'introduction de chacun des détails spécifiés.

  La spécification de la communication par APDU (très technique) a provoqué un fort accroissement de la taille de la spécification, puisque le modèle est passé en une étape de 350 lignes à 900 lignes de code JML. À ce niveau de détail, la taille de la spécification devient comparable à celle d'une implémentation telle que celle de Trusted Logic, qui est constituée de 1300 lignes de code Java.

  Le modèle a pu être validé et animé au moyen de l'outil JML-TT animator. La preuve de la correction du raffinement d'un niveau à l'autre nécessite encore un outil de génération des obligations de preuve du raffinement.

Nous envisageons dans les mois prochains de valoriser le travail GECCOO autour de Demoney par la production d'un document de synthèse.

3.3 Propriétés de sécurité

Nous avons effectué un inventaire des propriétés de sécurité, basé sur des documents fournis par des industriels de la carte à puce. Nous distinguons deux niveaux de sécurité : le niveau le plus haut qui décrit des propriétés générales et le niveau inférieur qui identifie des règles concrètes que l'implémentation doit respecter. Le lien entre ces deux niveaux relève de l'analyse de sécurité; une partie de l'audit consiste à vérifier les règles imposées, ceci est souvent fait « à la main ».

Plusieurs difficultés se présentent, tout d'abord fournir des langages de haut niveau pour spécifier les propriétés de sécurité attendues, ensuite donner des moyens de garantir mécaniquement l'adéquation du code à la politique attendue.

3.3.1 Langage

L'équipe Vasco à Grenoble a proposé un langage de description de propriétés pour Génésyst à partir des travaux de l'équipe Lemme et l'a expérimenté sur l'application Demoney ([D.5]). Elle développe actuellement un langage de description de différentes formes de politiques de contrôle d'accès (politique discrétionnaire, politique obligatoire multi-niveaux, politique basée rôle) qui permet de produire la spécification des comportements attendus au niveau d'une description fonctionnelle du système (outil MECA [D.43]).

3.3.2 Propriétés temporelles en Jml

Ce travail est réalisé en collaboration par l'équipe TFC à Besançon et le projet Everest à Sophia-Antipolis.

Un langage de logique temporelle pour JML, appelé JTPL (Java Temporal Pattern Language), a été proposé par Huisman et Trentelman [D.63]. Dans ce travail, une méthode de génération automatique d'annotations JML à partir des formules exprimant des propriétés de sûreté a été proposée. Nous avons étendu ce travail pour des propriétés de vivacité en proposant des obligations de preuves traduisibles en JML standard qui assurent la vérification des propriétés de vivacité exprimables dans ce langage. La correction des annotations générées assure la satisfaction des propriétés de vivacité pour une classe C utilisée par un environnement générique défini par une hypothèse de progrès. Ce travail a abouti à la rédaction d'un rapport technique INRIA [E.2] et à une communication à la conférence SAVCB 2006 [D.40].

La seconde phase du travail consiste à vérifier la préservation des propriétés de vivacité lorsque la classe C est utilisée par un système donné. Plus précisément, cette seconde phase consiste à montrer – en utilisant des règles spéciales de Hoare de *correction partielle avec progrès* – que le programme préserve les propriétés temporelles établies pour C, i.e. que le programme satisfait l'hypothèse de progrès. Ce travail est en cours de rédaction dans un article soumis en revue. Nous avons montré la faisabilité de cette étape en utilisant une technique de traduction en B. Ce travail a été publié à la conférence B 2005 [D.11]. Une solution meilleure a été proposée en produisant des obligations de preuve qui impliquent l'hypothèse de progrès et directement déchargées par Krakatoa ou Jack. Enfin, nous avons également proposé une méthode de génération automatique de tests de propriétés de sécurité exprimées en JTPL en utilisant JML-TT. Ce travail a été publié à FATES 2006 [D.10]. Ces travaux sont implantés dans l'outil JAG (cf section 3.1.1).

Nous avons montré que la démarche initiale de JAG n'est pas limitée à JML et au langage JTPL. Une communication à la conférence B'07 étend la démarche à l'ensemble des propriétés temporelles exprimées par des automates de Buchi [D.42] dans le cadre des systèmes d'événements B. Cette extension a été intégrée à l'outil JAG [D.41].

3.3.3 Automates finis vers Jml

Les propriétés de sécurité peuvent être spécifiées par des machines à états finies. L'équipe Everest a montré comment engendrer des annotations JML à partir d'une telle spécification. L'algorithme est implanté en Jack et est en cours de formalisation. Étant donné une propriété décrite par un automate fini et une application JAVA, une sémantique de «monitoring » est définie qui décrit comment les transitions de l'automates peuvent être déclenchées par des appels de méthode JAVA. L'automate est étendu de manière à ce que qu'il aboutisse dans un état-puits s'il ne peut pas exécuter de transition. On ajoute un invariant au programme JAVA comme quoi l'état puits n'est jamais atteint. On montre que si la sémantique de monitoring n'atteint pas l'état-puits alors l'invariant n'est pas violé. Pour l'instant on montre qu'il n'y aura pas d'erreur dynamique, l'objectif étant de propager les annotations pour effectuer une vérification statique.

3.4 Modularité et raffinement

3.4.1 Invariants

Un aspect important est l'étude de la notion d'invariant de classes (ou de modules) et la possibilité de modulariser les preuves d'invariant. Ces aspects s'avèrent essentiels pour pouvoir vérifier des applications de taille conséquente tout en maîtrisant les tâches de spécification et de vérification formelle

et aussi pour pouvoir réutiliser et assembler des spécifications. Dans sa forme actuelle la méthode B offre un jeu de primitives de structuration qui permet de maîtriser finement la composition d'invariants, et les preuves associées. Dans la pratique, les règles imposées par ce jeu de primitives s'avèrent trop restrictives et obligent les spécifieurs soit à construire des architectures non nécessairement très naturelles soit à contourner d'une manière ou d'une autre ces restrictions. Dans les approches objet d'autres solutions sont adoptées, avec leurs propres difficultés et restrictions. Les résultats obtenus sont les suivants:

- Analyse de la notion d'invariant dans différents langages (B, JML et Spec#) et comparaison (Dagsthul Seminar 06191 « Rigorous Methods for Software Construction and Analysis »[2]).
- Validation de la correction de la méthode B et proposition d'extension de cette méthode en utilisant les notions introduites dans l'approche Spec#[3] (statut des composants et relation d'appartenance) [D.9].
- Etude de patrons de spécification permettant aux spécifieurs de maîtriser la complexité des preuves d'invariant et de rester compatible avec un principe de substitutivité par raffinement (travaux en cours). Ces travaux ont aussi donné lieu à des collaborations avec l'équipe du LRI: en particulier une proposition d'interprétation des invariants de classe en présence de redéfinition a été élaborée.

3.4.2 Raffinement et sécurité

Un objectif du projet GECCOO était de définir des conditions de préservation par raffinement des propriétés de sécurité.

Différentes équipes ont travaillé sur ce thème. L'équipe de Besançon a travaillé sur la préservation de propriétés temporelles par raffinement (voir section 3.3.2). Le projet Vasco s'est intéressé au raffinement des données en lien avec les exigences de sécurité. En effet, dans le cas de politiques de contrôle d'accès une des difficultés est de manipuler des politiques à différents niveaux de spécifications (utilisateurs, pare-feux ...). Nous avons proposé une approche permettant de construire de manière systématique, par raffinement, des moniteurs de surveillance dans le cas des réseaux [D.60, D.61]. Dans cette approche les données manipulées par les politiques sont raffinées et l'approche proposée permet de garantir la préservation de la politique abstraite. Ce travail a été réalisé en collaboration avec l'ACI Potestat.

3.4.3 Raffinement et ordre supérieur

Des études ont été menées pour développer un calcul du raffinement objet, en étendant l'approche B. Étendre le raffinement à la programmation OO suppose de pouvoir prendre en compte les fonctions impératives d'ordre supérieur (un objet étant une valeur du langage qui « transporte » des méthodes effectuant des effets de bords). S. Boulmé a décrit en COQ un calcul de raffinement d'ordre supérieur et proposé une méthodologie pour prouver les fonctions impératives d'ordre supérieur via cette notion de raffinement [E.3].

Le thème d'étendre les logiques de Hoare avec des fonctions d'ordre supérieur est à la mode. En particulier, K. Honda, N. Yoshida et M. Berger[4] ont proposé une logique complète. Par rapport à ce travail, nous supportons « moins » de fonctions d'ordre supérieur. Mais notre proposition se situe dans le cadre du raffinement: une extension de la logique de Hoare qui a prouvé son intérêt dans les gros développements industriels.

[2] http://kathrin.dagstuhl.de/06191/Materials2/Dagsthul

[3] K.R.M. Leino and P. Müller, Object invariants in dynamic contexts, ECOOP, LNCS 3086, 2004

[4] An Observationally Complete Logic for Imperative Higher-Order Functions, LICS'05

3.4.4 Spécifications abstraites

Le langage de spécification JML a été développé en particulier pour permettre la génération de tests dynamiques, il favorise donc des spécifications calculatoires exprimées sous formel de code JAVA sans effet de bord. Cette approche n'est pas toujours la plus adaptée pour faire de la preuve de programme. Les projets Everest et ProVal ont rapidement rencontré ce problème et ont proposé des solutions pour les outils Jack et Krakatoa.

J. Charles du projet Everest propose l'introduction dans JML d'un mot clé `native` pour connecter une spécification JML avec la logique du prouveur sous-jacent, et permettre ainsi de raisonner sur des structures de données complexes [D.21].

C. Marché et N. Rousset du projet ProVal ont quant à eux proposé une notion de spécification de modèles logiques (à l'aide de déclarations de types, prédicats, fonctions et axiomes) ainsi que des mécanismes permettant de lier les données du programme à un modèle et de spécifier le comportement du programme par rapport aux opérations du modèle. Les liens entre cette méthode et la notion de raffinement restent à approfondir.

3.5 Génération et résolution d'obligations de preuves

3.5.1 Génération d'obligations de preuves

Une difficulté rencontrée pour le traitement d'exemples significatifs est d'améliorer la génération d'obligations de preuves. Celles-ci doivent en effet, autant que possible, pouvoir être traitées par des démonstrateurs automatiques.

Nous avons, au court du projet, effectué plusieurs adaptations du générateur d'obligations de preuve de Why qui sert de base à l'outil Krakatoa afin d'améliorer cette efficacité.

Sur le plan conceptuel, la nature des obligations de preuve est liée au modèle mémoire utilisé pour décrire la sémantique du programme JAVA. Le modèle décrit dans [C.8] a été raffiné pour prendre en compte la séparation mémoire des champs dans les représentations objet [D.49]. Ce traitement est insuffisant, des travaux menés dans le cadre de l'analyse de programmes C ont permis d'augmenter la séparation par l'introduction de zones détectées par typage. Ces techniques qui ont donné d'excellents résultats sur des codes industriels sont en cours de transfert pour les programmes JAVA.

Modèle pour JavaCard Un autre avantage du modèle paramétrable de Krakatoa est de pouvoir s'adapter aux spécificités d'un langage.

N. Rousset a développé un plugin de Krakatoa pour les programmes JAVACARD. Il distingue la mémoire volatile et la mémoire persistante et prend en compte le mécanisme de transactions qui permet de considérer un ensemble d'opérations comme devant se dérouler de manière atomique. Les cartes à puce sont sensibles à l'arrachement, et il faut garantir la cohérence des données lorsque la carte est remise en route. Le langage de spécification de JML a été étendu pour prendre en compte le comportement attendu en cas d'arrachement, le mécanisme d'exception de Why est utilisé pour raisonner sur les cas d'arrachement. L'exemple du PIN code montre que la combinaison du mécanisme de transaction et d'arrachage peut introduire des trous de sécurité. Notre modélisation a permis de détecter une attaque dans une implantation naïve de la vérification de code PIN et a permis de prouver sûre une version corrigée. Ce travail a été publié à la conférence SEFM'06 [D.50].

3.5.2 Techniques de démonstration

Le travail sur les techniques de démonstration automatique s'oriente dans plusieurs directions.

– La preuve de programmes nécessite de combiner différentes théories. On sait que c'est un problème difficile en général et les résultats connus imposent des restrictions sur les différentes sous-théories. Le projet CASSIS [D.62, D.35, D.57, C.11] a obtenu des résultats pour relâcher certaines de ces conditions. Il s'intéresse plus particulièrement à la combinaison de théories sur les listes, les ensembles et les théories sous-jacentes sur les éléments.
– Il est possible de trouver des théories décidables pour certaines classes de propriétés à vérifier sur les programmes. Une expérience intéressante a été menée pour montrer des propriétés d'accessibilité de certaines parties de la mémoire d'un programme de GC [D.51].
– Les démonstrateurs automatiques sont des programmes complexes dont les capacités sont limitées. Il est important de pouvoir combiner leur utilisation avec des outils de démonstration interactive. Une technique est de revérifier a posteriori des traces de preuve trouvées automatiquement.

Extensions de la théorie des tableaux Parmi les théories intéressantes pour l'analyse de programme, on note la théorie des ensembles avec cardinalité. S. Ranise en collaboration avec S. Ghilardi, E. Nicolini et D. Zucchelli a exploré la théorie des tableaux augmentés par des variables entières qui comptent le nombre d'éléments stockés pour chaque valeur. Dans [D.36], il s'est intéressé à la combinaison de cette théorie des tableaux augmentés avec une théorie dans laquelle les termes d'indice sont dans l'arithmétique de Pressburger et où des symboles de fonctions et de prédicats supplémentaires sont introduits (dimension du tableau, propriété d'être trié). Le résultat principal est de ramener la satisfiabilité de l'union de ces deux théories à l'utilisation de procédures de décision pour chacune des théories. Ces travaux sont similaires à ceux de Bradley et al mais permettent de traiter des tableaux injectifs. La stratégie d'instanciation basée sur la superposition se rapproche de celle de Korovin et Ganzinger mais permet d'éviter le problème d'explosion combinatoire.

Types de données finis Il est difficile de combiner les sortes finies qui apparaissent naturellement dans les programmes (types énumérés) et des théories ayant des modèles infinis. En particulier les théories finies ne satisfont pas l'hypothèse de stabilité infinie requise dans la méthode de combinaison dûe à Nelson-Oppen.

Dans [D.7], S. Ranise en collaboration avec M. Bonacina, S. Ghilardi, E. Nicolini et D. Zucchelli, étudie le problème de la décidabilité de la satisfiabilité dans une théorie, en distinguant les modèles finis et infinis. Il montre en particulier que la combinaison d'une théorie décidable T_1 (mais indécidable dans les modèle infinis) et d'une théorie décidable T_2 peut être indécidable même avec des symboles disjoints. Il montre également comment combiner des théories lorsque la satisfiabilité s'appuie sur des techniques de réécriture et permet de combiner des techniques par réécriture et la résolution de contraintes pour les modèles finis.

Pointeurs La vérification de programmes requiert de raisonner sur la mémoire et les structures de pointeurs en particulier l'accessibilité qui n'est pas une définition du premier ordre.

S. Ranise et C. Zarba [D.58] ont proposé une théorie des listes chaînées qui permet de raisonner sur les cellules (donnée et pointeur), sur des collections de cellules et sur l'accessibilité d'une cellule à partir d'une autre. Ils ont montré que le problème de décision de cette théorie était NP-complet mais qu'il était possible de combiner cette théorie avec des théories décidables sur les éléments ou les pointeurs en suivant le schéma [D.57].

De nombreux résultats existent dans ce domaine. Mehta et Nipkow utilisent Isabelle/HOL pour résoudre par tactique les obligations de preuve associées aux programmes manipulant des listes chaînées. Des procédures (Balaban et al, Bingham et al) ont été proposées dans le cadre de la logique de séparation mais en abstrayant la structure des données et des pointeurs. Nelson, McPeak

et Necula d'une part, Qaader et Lahiri d'autres part proposent des solutions qui combinent analyse d'accessibilité et décision sur les données mais avec une notion d'accessibilité restreinte au premier ordre.

CC(X) E. Contejean et S. Conchon ont entrepris l'implantation d'un nouveau prouveur Ergo qui s'appuie sur une méthode optimisée de Congruence Closure modulo une théorie X. L'implantation est très courte et suit la description logique du système ce qui en fait un bon candidat pour la certification. L'implantation de Ergo a nécessité la mise en place d'algorithmes fonctionnels efficaces pour le backtracking [D.34], le hash-consing [D.22] et les structures Union-Find [D.23].

Interactions de différents prouveurs Le projet ProVal a travaillé sur la possibilité de faire interagir différents démonstrateurs pour résoudre les obligations de preuve. Ce travail a donné lieu à plusieurs résultats.

- Une méthode générale d'appel de prouveurs automatiques du premier ordre à partir de l'assistant Coq [E.1]. Le problème principal est d'interpréter au premier ordre une grande partie du contexte du but qui est écrit dans une logique d'ordre supérieur. La tactique correspondante a été intégrée à Coq et fonctionne en particulier avec Simplify et Ergo. Pour l'instant, la propriété prouvée automatiquement est vue comme un axiome dans Coq.
- Des techniques de reconstruction de termes de preuve pour Coq à partir d'outils automatiques. Dans [D.24], P. Corbineau et E. Contejean utilisent un langage de traces général pour la logique du premier ordre avec égalité afin de traduire des preuves de l'outil CiMe vers Coq.
- La logique de Why est une logique multi-sortée polymorphe. Il convient de traduire les obligations de preuve vers des démonstrateurs du premier ordre parfois non sortée. C'est un problème non trivial si on veut préserver la correction et l'efficacité. Nous avons proposé et implanté plusieurs méthodes qui sont en cours d'évaluation [E.8].

3.6 Détection d'erreurs

Ces travaux ont été réalisés à Besançon, ils apportent des méthodes et outils nouveaux pour traiter des spécifications Jml: l'animation, la vérification de propriétés sur les spécifications et la génération de tests à partir des spécifications.

Animation Le premier travail que nous avons effectué dans le projet est de réaliser les extensions nécessaires à l'approche BZ-Testing-Tools pour prendre en compte les concepts de la notation Jml.

Pour ce faire, nous sommes partis d'un environnement d'animation et de résolution de contraintes logico-ensembliste [D.18], initialement conçu pour la prise en compte de spécifications B, et basé sur un format intermédiaire nommé BZP. Nous avons proposé une représentation des états mémoire Java dans ce format ensembliste, permettant ainsi leur expression sous la forme de systèmes de contraintes. Nous avons ensuite défini l'expression de spécifications Jml dans le format BZP, qui nous a permis de pratiquer l'animation de la spécification [D.14, D.13].

Mise au point de spécification Dans le cadre de la vérification de programmes basée sur la modélisation JML, il est important de s'assurer que le modèle Jml de l'application ne comporte pas d'incohérence. Bien qu'il existe de nombreux outils de vérification de la correction du code Java vis-à-vis de la spécification JML, il n'existait pas d'outil permettant de s'assurer qu'il n'y a pas d'incohérence dans le modèle Jml lui-même. Nous avons proposé une technique de vérification basée sur une traduction de la spécification JML vers le langage des machines abstraites B [D.11]. La cohérence de la machine B générée, vérifiée avec l'Atelier B, implique la cohérence du modèle Jml

original. Ce travail a été publié à la conférence ZB'2005 [D.11] et a donné lieu à l'implantation du prototype JML2B [D.12]. Celui-ci a permis de vérifier la cohérence du modèle JML de Demoney utilisé pour la génération de tests [D.10].

Génération de tests La dernière étape du travail concerne la génération de séquence de tests [D.29]. Nous avons étendu la notion de génération de tests aux limites pour les programmes Java à partir de leur spécification JML. Nous avons ainsi défini deux types d'objectifs de tests. Le premier concerne l'activation des comportements extraits des spécifications JML des méthodes [D.17]. La seconde concerne l'utilisation de propriétés de sécurité comme objectif de tests [D.10]. Nous avons défini différents critères de couverture à partir de la spécification. Nous avons à cette occasion défini la notion de « valeur aux limites » pour des objets.

4 Conclusion

4.1 Visibilité

L'équipe TFC à Besançon a organisé les journées « Approches Formelles dans l'Assistance au Développement de Logiciels » (AFADL'2004) en juin 2004 et organise la « 7th International B Conference » (B'2007) en janvier 2007. Ces deux manifestation ont permis la présentation de travaux issus du projet GECCOO.

C. Marché a été invité à présenter ses travaux au workshop « Challenges in Java Program Verification » `http://www.cs.ru.nl/~woj/esfws06/`.

S. Ranise coordonne avec Cesare Tinelli l'effort de construction d'une bibliothèque `SMT-lib` autour du problème de satisfiabilité modulo une théorie [F.5, E.9]. La bibliothèque regroupe des benchmarks, des utilitaires (elle propose un standard d'entrée) et des outils.

4.2 Apports du projet

Le projet a permis de favoriser des échanges entre des outils de même nature (les outils Jack et Krakatoa d'analyse de code JAVA, les démonstrateurs automatiques haRVey ou ERGO) ou complémentaires (générateur d'obligations et prouveur). Il a permis également de transposer et adapter des technologies de la méthode B vers le formalisme JAVA/JML qui est plus adapté aux chaînes de développement de logiciel dans l'industrie (JML-TESTING-TOOLS, JML2B). Il contribue à prendre en compte dans les outils des spécifications de sécurité de haut niveau qui peuvent être mécaniquement vérifiées.

Le projet s'organise autour de recherches amont sur le raffinement pour des langages d'ordre supérieur avec effets de bord, ou la combinaison de procédures de décision; mais aussi de développement d'outils et d'expérimentation sur des études de cas.

4.3 Changements par rapport au projet initial

Le projet a réalisé un effort important de convergence des formalismes sur lesquels se basent les méthodes et les outils. Les outils proposés permettent déjà de développer des expérimentations intéressantes. Cependant, comme indiqué dans le rapport à mi-parcours, l'objectif initial de construire un environnement unique allant de la spécification de haut niveau jusqu'à la simulation et le test n'a pas été atteint à la fin du projet. Un tel environnement réclamait un effort d'ingénierie trop important par rapport aux ressources du projet. Il a été plus efficace de se concentrer sur un petit nombre d'outils pouvant communiquer entre eux.

Livres et monographies

[A.1] J. Julliand, O. Kouchnarenko (réd.), *The 7th International B Conference (B'2007)*, *LNCS*, *4355*, Springer-Verlag, 2007.

Thèses

[B.1] F. Dadeau, *Évaluation symbolique à contraintes pour la validation, application à Java/JML*, thèse de doctorat, Université de Franche-Comté, 2006.

[B.2] M. Pavlova, *Bytecode Verification and its Applications*, thèse de doctorat, Université de Nice Sophia-Antipolis, janvier 2007.

Articles et chapitres de livre

[C.1] A. A, S. Ranise, M. Rusinowitch, A rewriting approach to satisfiability procedures, *Information and Computation 183*, 2, 2003, p. 140–164.

[C.2] A. Armando, L. Compagna, S. Ranise, Rewriting and Decision Procedure Laboratory: Combining Rewriting, Satisfiability Checking, and Lemma Speculation, *in : Mechanizing Mathematical Reasoning. Essays in Honor of J. H. Siekmann on the Occasion of his 60th Birthday*, D. Hutter et W. Stephan (réd.), *LNAI*, *2605*, Springer-Verlag, 2005, p. 30–45.

[C.3] F. Bellegarde, C. Charlet, O. Kouchnarenko, How to Compute the Refinement Relation for Parameterized Systems, *in : Formal Methods and Models for System Design*, R. Gupta, L. G. P., et T. J.P. (réd.), Springer-Verlag, 2004, ch. 2.

[C.4] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Ranise, R. Sebastiani, Efficient Theory Combination via Boolean Search, *Journal of Information and Computation 10*, 204, 2006, p. 1411–1596, Special Issue on Combining Logical Systems.

[C.5] C.-B. Breunesse, N. C. o, M. Huisman, B. Jacobs, Formal Methods for Smart Cards: an experience report, *Science of Computer Programming 55*, 1-3, 2005, p. 53–80.

[C.6] S. Chouali, J. Julliand, P.-A. Masson, F. Bellegarde, PLTL Partitionned Model-Checking for Reactive Systems under Fairness Assumptions, *ACM - Transactions in Embedded Computing Systems (TECS) 4*, 2, 2005, p. 267–301.

[C.7] J.-F. Couchot, D. Déharbe, A. Giorgetti, S. Ranise, Scalable Automated Proving and Debugging of Set-Based Specifications, *Journal of the Brazilian Computer Society 9*, 2, November 2003, p. 17–36, ISSN 0104-6500.

[C.8] C. Marché, C. Paulin-Mohring, X. Urbain, The Krakatoa Tool for Certification of Java/JavaCard Programs annotated in JML, *Journal of Logic and Algebraic Programming 58*, 1–2, 2004, p. 89–106.

[C.9] S. Ranise, C. Tinelli, Satisfiability Modulo Theories, *IEEE Magazine on Intelligent Systems*, November 2006, To appear.

[C.10] C. Tinelli, C. G. Zarba, Combining non-stably infinite theories, *Journal of Automated Reasoning 34*, 3, 2005, p. 209–238.

[C.11] C. G. Zarba, D. Cantone, J. T. Schwartz, A decision procedure for a sublanguage of set theory involving monotone, additive, and multiplicative functions, I. The: two-level case, *Journal of Automated Reasoning 33*, 3-4, 2004, p. 251–269.

Communications à des congrès, colloques, etc.

[D.1] A. Armando, M. P. Bonacina, S. Ranise, S. Schulz, Big proof engines as little proof engines: new results on rewrite-based satisfiability procedures (Extended abstract), *in : Notes of the Third Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR), co-located with CAV'05*, Edinburgh, Scotland (UK), 2005.

[D.2] A. Armando, M. P. Bonacina, S. Ranise, S. Schulz, On a rewriting approach to satisfiability procedures: extension, combination of theories and an experimental appraisal, *in : Proc. of the 5th Int. Workshop on Frontiers of Combining Systems (FroCoS'2005)*, B. Gramlich (réd.), *Lecture Notes in Computer Science*, *3717*, Springer, p. 65–80, Vienna, Austria, September 2005.

[D.3] G. Barthe, P. R. D'Argenio, T. Rezk, Secure Information Flow by Self-Composition, *in : 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, R. Foccardi (réd.), IEEE Computer Society, p. 100–114, Los Alamitos, CA, USA, June 2004.

[D.4] G. Barthe, M. Pavlova, G. Schneider, Precise analysis of memory consumption using program logics, *in : Proceedings of SEFM'05*, B. Aichernig, B. Beckert (réd.), IEEE Press, 2005.

[D.5] D. Bert, M.-L. Potet, N. Stouls, GeneSyst: a Tool to Reason about Behavioral Aspects of B Event Specifications. Application to Security Properties, *in : ZB 2005: Formal Specification and Development in Z and B, 4th International Conference of B and Z Users*, H. Treharne, S. King, M. C. Henson, S. A. Schneider (réd.), *LNCS*, *3455*, Springer-Verlag, p. 299–318, 2005.

[D.6] D. Bert, M.-L. Potet, N. Stouls, GénéSyst: a Tool to Reason about Behavioral Aspects of B Event Specifications. Application to Security Properties, *in : Proceedings of the International Conference on Formal Specification and Development in Z and B (ZB'05)*, *LNCS*, *3455*, Springer-Verlag, p. 299–318, Guildford, United Kingdom, avril 2005.

[D.7] M. P. Bonacina, S. Ghilardi, E. Nicolini, S. Ranise, D. Zucchelli, Decidability and Undecidability Results for Nelson-Oppen and Rewrite-Based Decision Procedures, *in : Proc. of the 3rd Int. Conference on Automated Reasoning (IJCAR)*, *LNAI*, 4130, p. 513–527, Seattle, WA, USA, August 2006.

[D.8] N. Boughanmi, S. Ranise, C. Ringeissen, On Structural Information and the Experimental Evaluation of SMT Tools, *in : Proc. of the International Workshop on First-Order Theorem Proving (FTP'05)*, Koblenz, Germany, 2005.

[D.9] S. Boulmé, M.-L. Potet, Interpreting invariant composition in the B method using the Spec# ownership relation: a way to explain and relax B restrictions, in : Julliand et Kouchnarenko [A.1].

[D.10] F. Bouquet, F. Dadeau, J. Groslambert, J. Julliand, Safety Property Driven Test Generation from JML Specifications, *in : FATES/RV'06, 1st Int. Workshop on Formal Approaches to Testing and Runtime Verification*, *LNCS*, *4262*, Springer, p. 225–239, Seattle, WA, USA, août 2006.

[D.11] F. Bouquet, F. Dadeau, J. Groslambert, Checking JML Specifications with B Machines, *in : Proceedings of the International Conference on Formal Specification and Development in Z and B (ZB'05)*, *LNCS*, *3455*, Springer-Verlag, p. 435–454, Guildford, United Kingdom, avril 2005.

[D.12] F. Bouquet, F. Dadeau, J. Groslambert, JML2B: Checking JML specifications with B machines, in : Julliand et Kouchnarenko [A.1].

[D.13] F. Bouquet, F. Dadeau, B. Legeard, M. Utting, JML-Testing-Tools: a Symbolic Animator for JML Specifications using CLP, *in : Proceedings of 11th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, Tool session (TACAS'05)*, *LNCS*, *3440*, Springer-Verlag, p. 551–556, Edinburgh, United Kingdom, April 2005.

[D.14] F. Bouquet, F. Dadeau, B. Legeard, M. Utting, Symbolic Animation of JML Specifications, *in : Proceedings of the International Conference on Formal Methods (FM'2005)*, *LNCS*, *3582*, Springer-Verlag, p. 75–90, Newcastle Upon Tyne, United Kingdom, July 2005.

[D.15] F. Bouquet, F. Dadeau, B. Legeard, How Symbolic Animation can help designing an Efficient Formal Model, *in : Procs of the 7th Int. Conf. on Formal Engineering Methods (ICFEM'05)*, *LNCS*, *3785*, Springer-Verlag, p. 96–110, Manchester, UK, novembre 2005.

[D.16] F. Bouquet, F. Dadeau, B. Legeard, Using Constraint Logic Programming for the Symbolic Animation of Formal Models, *in : Procs of the Int. Workshop on Constraints in Formal Verification (CFV'05) – Co-located with the Int. Conf. on Automated Deduction (CADE'05)*, J. Marques-Silva, M. Velev (réd.), p. 32–46, Tallinn, Estonia, juillet 2005.

[D.17] F. Bouquet, F. Dadeau, B. Legeard, Automated Boundary Test Generation from JML Specifications, *in : FM'06, 14th Int. Conf. on Formal Methods*, *LNCS*, *4085*, Springer-Verlag, p. 428–443, Hamilton, Canada, août 2006.

[D.18] F. Bouquet, F. Dadeau, B. Legeard, JML-Testing-Tools, un Animateur Symbolique de Spécifications JML, *in : AFADL'06, Approches Formelles dans l'Assistance au Développement de Logiciels*, Paris, France, mars 2006. Session outils.

[D.19] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Ranise, R. Sebastiani, Efficient Satisfiability Modulo Theories via Delayed Theory Combination, *in : Proc. of the Int. Conf. on Computer-Aided Verification (CAV 2005)*, *Lecture Notes in Computer Science*, 3576, Springer-Verlag, Edinburg, Scotland (UK), 2005.

[D.20] L. Burdy, M. Pavlova, Java Bytecode Specification and Verification, *in : 21st Annual ACM Symposium on Applied Computing (SAC'06)*, L. Liebrock (réd.), ACM Press, Dijon, avril 2006.

[D.21] J. Charles, Adding Native Specifications to JML, *in : Proceedings of the ECOOP workshop on Formal Techniques for Java-like Programs (FTfJP'2006)*, 2006.

[D.22] S. Conchon, J.-C. Filliâtre, Type-Safe Modular Hash-Consing, *in : ACM SIGPLAN Workshop on ML*, Portland, Oregon, septembre 2006.

[D.23] S. Conchon, J.-C. Filliâtre, Union-Find Persistant, *in : Dix-huitièmes Journées Francophones des Langages Applicatifs*, INRIA, janvier 2007. to appear.

[D.24] E. Contejean, P. Corbineau, Reflecting Proofs in First-Order Logic with Equality, *in : 20th International Conference on Automated Deduction (CADE-20)*, *LNAI*, 3632, Springer-Verlag, p. 7–22, Tallinn, Estonia, juillet 2005.

[D.25] J.-F. Couchot, F. Dadeau, D. Déharbe, A. Giorgetti, S. Ranise, Proving and Debugging Set-Based Specifications, *in : Electronic Notes in Theoretical Computer Science, proceedings of the Sixth Brazilian Workshop on Formal Methods (WMF'03)*, A. Cavalcanti, P. Machado (réd.), *95*, p. 189–208, Campina Grande, Brazil, May 2004.

[D.26] J.-F. Couchot, D. Déharbe, A. Giorgetti, S. Ranise, Barvey : Vérification automatique de consistance de machines abstraites B, *in : Sessions Outils, Congrès Approches Formelles dans l'Assistance au Développement de Logiciels, AFADL'04*, J. Julliand (réd.), p. 369–372, Besançon, France, juin 2004.

[D.27] J.-F. Couchot, A. Giorgetti, Analyse d'atteignabilité déductive, *in : Congrès Approches Formelles dans l'Assistance au Développement de Logiciels, AFADL'04*, J. Julliand (réd.), p. 269–283, Besançon, France, juin 2004.

[D.28] A. Courbot, M. Pavlova, G. Grimaud, J. Vandewalle, A Low-Footprint Java-to-Native Compilation Scheme Using Formal Methods., *in : Smart Card Research and Advanced Applications, 7th IFIP WG 8.8/11.2 International Conference, CARDIS 2006*, J. Domingo-Ferrer, J. Posegga, D. Schreckling (réd.), *LNCS*, *3928*, Springer-Verlag, p. 329–344, avril 2006.

[D.29] F. Dadeau, Animation de modèles JML et génération de tests fonctionnels, *in : MAJECSTIC'06, MAnifestation de JEunes Chercheurs STIC*, Lorient, France, novembre 2006.

[D.30] D. Déharbe, P. Fontaine, S. Ranise, C. Ringeissen, Decision Procedures for the Formal Analysis of Software, *in : 3rd International Colloquium on Theoretical Aspects of Computing, ICTAC*, *Lecture Notes in Computer Science*, *4281*, Springer, Tunis, Tunisia, November 2006. Tutorial.

[D.31] D. Déharbe, S. Ranise, Light-Weight Theorem Proving for Debugging and Verifying Units of Code, *in : Proc. of the International Conference on Software Engineering and Formal Methods (SEFM03)*, IEEE Computer Society Press, Brisbane, Australia, September 2003.

[D.32] D. Déharbe, S. Ranise, Satisfiability Solving for Software Verification, *in : ISoLA-2005: 2005 IEEE ISoLA Workshop on Leveraging Applications of Formal Methods, Verification, and Validation*, Columbia, Maryland, 2005.

[D.33] J.-C. Filliâtre, C. Marché, Multi-Prover Verification of C Programs, *in : Sixth International Conference on Formal Engineering Methods*, J. Davies, W. Schulte, M. Barnett (réd.), *LNCS*, *3308*, Springer-Verlag, p. 15–29, Seattle, WA, USA, novembre 2004.

[D.34] J.-C. Filliâtre, Backtracking iterators, *in : ACM SIGPLAN Workshop on ML*, Portland, Oregon, septembre 2006.

[D.35] P. Fontaine, S. Ranise, C. G. Zarba, Combining lists with non-stably infinite theories, *in : Logic for Programming, Artificial Intelligence, and Reasoning*, F. Baader, A. Voronkov (réd.), *Lecture Notes in Computer Science*, *3452*, Springer-Verlag, p. 51–66, 2005.

[D.36] S. Ghilardi, E. Nicolini, S. Ranise, D. Zucchelli, Deciding Extensions of the Theory of Arrays by Integrating Decision Procedures and Instantiation Strategies, *in : Proc. of the 10th European Conference on Logics in Artificial Intelligence (Jelia)*, *Lecture Notes in Computer Science*, 4160, September 2006. Extended version with full proofs, motivations, and examples in a technical report available at `http://www.loria.fr/~ranise/pubs`.

[D.37] S. Ghilardi, E. Nicolini, S. Ranise, D. Zucchelli, Deciding Extensions of the Theory of Arrays by Integrating Decision Procedures and Instantiation Strategies, *in : Proc. of the IJCAR'06 Ws. PDPAR: Pragmatical Aspects of Decision Procedures in Automated Reasoning*, B. Cook, R. Sebastiani (réd.), Seattle, WA, USA, August 2006.

[D.38] A. Giorgetti, J. Groslambert, JAG : Génération d'annotations JML pour vérifier des propriétés temporelles, *in : AFADL'06, Approches Formelles dans l'Assistance au Développement de Logiciels*, Paris, France, March 2006. Session outils.

[D.39] A. Giorgetti, J. Groslambert, JAG: JML Annotation Generation for Verifying Temporal Properties, *in : FASE'2006, Fundamental Approaches to Software Engineering*, *LNCS*, *3922*, Springer, p. 373–376, Vienna, Austria, mars 2006.

[D.40] J. Groslambert, J. Julliand, O. Kouchnarenko, JML-based Verification of Liveness Properties on a Class, *in : SAVCBS'06, Specification and Verification of Component-Based Systems*, Portland, Oregon, USA, novembre 2006.

[D.41] J. Groslambert, A JAG extension for verifying LTL properties on B Event Systems, in : Julliand et Kouchnarenko [A.1], p. 262–265.

[D.42] J. Groslambert, Verification of LTL on B Event Systems, in : Julliand et Kouchnarenko [A.1].

[D.43] A. Haddad, Meca: a tool for Access Control Models, in : Julliand et Kouchnarenko [A.1]. Tool session.

[D.44] M. Huisman, P. Worah, K. Sunesen, A temporal logic characterisation of observational determinism, *in : 19th IEEE Computer Security Foundations Workshop*, IEEE Computer Society, July 2006.

[D.45] A. Imine, D. Déharbe, S. Ranise, Abstraction-Driven Verification of Array Programs, *in : Proceedings of the 7th International Conference on Artificial Intelligence and Symbolic Computation (AISC'04)*, *LNCS*, 3249, Springer-Verlag, p. 271–275, Linz, Austria, septembre 2004.

[D.46] H. Kirchner, S. Ranise, C. Ringeissen, D.-K. Tran, On Superposition-Based Satisfiability Procedures and their Combination, *in : Proc. of the 2nd International Conference on Theoretical Aspects of Computing (ICTAC'05)*, D. Van Hung, M. Wirsing (réd.), *Lecture Notes in Computer Science*, *3722*, Springer, p. 594–608, Hanoi, Vietnam, October 2005.

[D.47] H. Kirchner, S. Ranise, C. Ringeissen, D.-K. Tran, Automatic Combinability of Rewriting-Based Satisfiability Procedures, *in : Proc. of the 13th Int. Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'06)*, *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Phnom Penh, Cambodia, novembre 2006.

[D.48] H. Kirchner, S. Ranise, C. Ringeissen, D.-K. Tran, Building and Combining Satisfiability Procedures for Software Verification, *in : Proceedings of 3rd Taiwanese-French Conference on Information Technology (TFIT)*, p. 125–139, Nancy, France, March 2006.

[D.49] C. Marché, C. Paulin-Mohring, Reasoning about Java Programs with Aliasing and Frame Conditions, *in : 18th International Conference on Theorem Proving in Higher Order Logics*, J. Hurd, T. Melham (réd.), *LNCS*, Springer-Verlag, août 2005.

[D.50] C. Marché, N. Rousset, Verification of Java Card Applets Behavior with respect to Transactions and Card Tears, *in : 4th IEEE International Conference on Software Engineering and Formal Methods (SEFM'06)*, D. V. Hung, P. Pandya (réd.), Pune, India, septembre 2006.

[D.51] F. Mehta, S. Ranise, Automated Provers doing (Higher-Order) Proof search: A Case Study in the Verification of Pointer Programs, *in : Proc. of the 2nd Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR'04)*, 2004.

[D.52] X. Morselli, M.-L. Potet, N. Stouls, GénéSyst : Génération d'un système de transitions étiquetées à partir d'une spécification B événementiel, *in : AFADL'2004*, p. 317–320, juin 2004.

[D.53] M. Pavlova, G. Barthe, L. Burdy, M. Huisman, J.-L. Lanet, Enforcing High-Level Security Properties For Applets, *in : CARDIS'04*, J.-J. Quisquater, P. Paradinas, Y. Deswarte, A. E. Kalam (réd.), Kluwer, 2004. An earlier version appeared as INRIA Technical Report, nr. RR-5061.

[D.54] M.-L. Potet, N. Stouls, Explicitation du contrôle de développement B événementiel, *in : AFADL'2004*, J. Julliand (réd.), p. 13–27, juin 2004.

[D.55] S. Ranise, C. Ringeissen, D.-K. Tran, Nelson-Oppen, Shostak and the Extended Canonizer: A Family Picture with a Newborn, *in : First International Colloquium on Theoretical Aspects of Computing - ICTAC 2004*, K. Araki, Z. Liu (réd.), *LNCS*, Springer-Verlag, Guiyang, Chine, septembre 2004. In post-event proceedings. Also available as Research Report A04-R-193, LORIA, France.

[D.56] S. Ranise, C. Ringeissen, D.-K. Tran, Producing Conflict Sets for Combination of Theories, *in : Pragmatics of Decision Procedures in Automated Reasoning (PDPAR)*, B. Cook, R. Sebastiani (réd.), Seattle (WA), August 2006. Workshop affiliated to the 3rd International Joint Conference on Automated Reasoning, IJCAR.

[D.57] S. Ranise, C. Ringeissen, C. G. Zarba, Combining data structures with nonstably infinite theories using many-sorted logic, *in : Proc. of the 5th Int. Workshop on Frontiers of Combining Systems (FroCoS'2005)*, B. Gramlich (réd.), *Lecture Notes in Computer Science*, *3717*, Springer, p. 48–64, Vienna, Austria, September 2005.

[D.58] S. Ranise, C. Zarba, A Theory of Singly-Linked Lists and its Extensible Decision Procedure, *in : Proc. of the 4th IEEE International Conference on Software Engineering and Formal Methods (SEFM)*, IEEE Computer Society Press, Pune, India, September 2006. Extended version with full proofs, motivations, and examples in a technical report available at `http://www.loria.fr/~ranise/pubs`.

[D.59] S. Ranise, Satisfiability Solving for Program Verification: towards the efficient Combination of Automated Theorem Provers and Satisfiability Modulo Theory Tools, *in : Proc. of the IJCAR'06 Ws. DISPROVING: Non-Theorems, Non-Validity, Non-Provability*, W. Ahrendt, P. Baumgartner, H. de Nivelle (réd.), p. 49–58, Seattle, WA, USA, August 2006. Invited paper.

[D.60] N. Stouls, V. Darmaillacq, Développement formel d'un moniteur détectant les violations de politiques de sécurité de réseaux, *in : Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'06)*, S. Vignes, V. Donzeau-Gouge (réd.), p. 179–193, mars 2006. http://www-lsr.imag.fr/Les.Personnes/Nicolas.Stouls/Productions/2005-09-AFADL06/AFADL06.pdf.

[D.61] N. Stouls, M.-L. Potet, Security Policy Enforcement Through Refinement Process, p. 216–231, 2007.

[D.62] C. Tinelli, C. G. Zarba, Combining Decision Procedures for Sorted Theories, *in : Logics in Artificial Intelligence*, J. J. Alferes, J. A. Leite (réd.), *Lecture Notes in Computer Science*, *3229*, Springer-Verlag, p. 641–653, 2004.

[D.63] K. Trentelman, M. Huisman, Extending JML Specifications with Temporal Logic, *in : Algebraic Methodology And Software Technology (AMAST '02)*, *LNCS*, *2422*, Springer-Verlag, p. 334–348, 2002.

Rapports de recherche et publications internes

[E.1] N. Ayache, *Coopération d'outils de preuve interactifs et automatiques*, Mémoire, Université Paris 7, 2005.

[E.2] F. Bellegarde, J. Groslambert, M. Huisman, O. Kouchnarenko, J. Julliand, Verification of Liveness Properties with JML, *rapport de recherche*, INRIA, 2004.

[E.3] S. Boulmé, Specifying and reasoning in Coq with high-order impure programs using specifications a la Dijkstra and refinement, *rapport de recherche*, LSR laboratory, 2006, http://www-lsr.imag.fr/Les.Personnes/Sylvain.Boulme/horefinement.

[E.4] J. Charles, *Vérification d'un composant Java: Le vérificateur de bytecode*, Mémoire, Université de Nice, 2005.

[E.5] V. Chaudhary, The Krakatoa tool for certification of Java/JavaCard programs annotated in JML : A Case Study, *rapport de recherche*, IIT internship report, juillet 2004.

[E.6] F. Dadeau, A. Giorgetti, Vérification de machines abstraites B en logique monadique du second ordre, *Rapport de Recherche N° RR2003-01*, LIFC - Laboratoire d'Informatique de l'Université de Franche Comté, octobre 2003.

[E.7] A. Haddad, Modélisation et vérification de politiques de sécurité, *rapport de recherche*, LSR laboratory, 2005, http://www-lsr.imag.fr/Les.Personnes/Marie-Laure.Potet/PUBLI/DEA_Amal_HADDAD.pdf.

[E.8] S. Lescuyer, *Codage de la logique du premier ordre polymorphe multi-sortée dans la logique sans sortes*, Mémoire, MPRI, 2006.

[E.9] S. Ranise, C. Tinelli, The SMT-LIB Standard: Version 1.2, *rapport de recherche*, Department of Computer Science, The University of Iowa, 2006, Available at `www.SMT-LIB.org`.

[E.10] N. Stouls, Documentation d'introduction aux critères communs, *rapport de recherche*, LSR laboratory, 2004,
http://www-lsr.imag.fr/Les.Personnes/Nicolas.Stouls/Productions/CC/CriteresCommuns.pdf.

Divers

[F.1] G. Barthe, L. Burdy, J. Charles, B. Grégoire, M. Huisman, J.-L. Lanet, M. Pavlova, A. Requet, JACK: a tool for validation of security and behaviour of Java applications, 2006, Abstract for tutorial, longer version to appear.

[F.2] L. Burdy, J.-L. Lanet, A. Requet, JACK: Java Applet Correctness Kit, 2004, `http://www-sop.inria.fr/everest/soft/Jack/jack.html`.

[F.3] D. Déharbe, S. Ranise, haRVey, 2004, `http://www.loria.fr/~ranise/haRVey`.

[F.4] J.-C. Filliâtre, The Why certification tool, `http://why.lri.fr/`.

[F.5] S. Ranise, C. Tinelli, The Satisfiability Modulo Theories Library (SMT-LIB), `www.SMT-LIB.org`, 2006.