Action Concertée Incitative

SÉCURITÉ INFORMATIQUE

Descriptif complet du projet

Attention,
toute proposition doit faire l'objet d'un résumé enregistré
avant le 28 mars 2003
directement sur le site web de l'ACI :
http://aciSI.loria.fr

Chaque dossier doit être

– déposé avant le 04 avril 2003 sur le site web de l'ACI[1] :

http://aciSI.loria.fr

– envoyé par voie postale avec les signatures requises

avant le 11 avril 2003
(cachet de la poste faisant foi)

à

Ministère délégué à la Recherche et aux Nouvelles Technologies
Direction de la Recherche
Cellule ACI
ACI Sécurité Informatique
1, rue Descartes
75231 Paris cedex 05

[1] En cas de difficulté, une soumission par courrier électronique est possible à l'adresse aciSI@loria.fr

Action Concertée Incitative

SÉCURITÉ INFORMATIQUE

Descriptif complet du projet

I - FICHE D'IDENTITÉ DU PROJET

Nom du Projet : *(maximum 20 caractères)*

GECCOO

Titre du Projet : *(maximum 3 lignes)*

GECCOO : Génération de code certifié pour des applications orientées objet Spécification, raffinement, preuve et détection d'erreurs.

Type du Projet :

Projet de recherche	Projet de recherche multi-thématiques	Projet de recherche avec infrastructure	Autre
XXX			

Durée du projet : 36 mois

Description courte du Projet : *(une demi-page maximum)*

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). Les travaux récents ont montré l'intérêt de spécifier 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.

Cependant les premières expériences menées ont permis d'identifier plusieurs verrous que ce projet se propose d'attaquer. 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 auront un champ large d'application dans les environnements destinés à produire du code objet garanti correct par rapport à des spécifications formelles. Une attention particulière sera apportée à mettre en œuvre ces techniques dans les outils développés par les partenaires du projet.

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). En effet, la réussite de la chaîne de développement que nous voulons mettre en place dépend fortement de la bonne articulation des différents maillons.

Coordinateur du projet :

Nom	Prénom	Laboratoire (sigle éventuel et nom complet)
PAULIN	Christine	LRI-Laboratoire de Recherche en Informatique

Organisme de rattachement financier pour le présent projet :

INRIA Futurs

Équipes ou laboratoires partenaires (nom complet et éventuellement sigle)

Équipe TFC Laboratoire d'informatique de Franche-Comte (LIFC) FRE 2661 CNRS Université de Franche-Comté, Besançon
Projet CASSIS[2] Laboratoire Lorrain de Recherche en Informatique et ses Applications (LORIA) UMR 7503, CNRS - INPL - INRIA - UHP, Nancy 1 - Nancy 2
Projet Lemme INRIA Sophia-Antipolis
Projet LogiCal Laboratoire de Recherche en Informatique (LRI) UMR 8623 CNRS, Université Paris Sud, Orsay et INRIA Futurs, Saclay
Équipe VASCO Laboratoire Logiciels Systèmes Réseaux (LSR)-UMR 5526 CNRS, INPG, UJF, Grenoble

[2]Le projet CASSIS est commun avec le Laboratoire d'informatique de Franche-Comte

Action Concertée Incitative

SÉCURITÉ INFORMATIQUE

Descriptif complet du projet

II - PRÉSENTATION DÉTAILLÉE DU PROJET

A - IDENTIFICATION DU COORDINATEUR ET DES AUTRES PARTENAIRES DU PROJET :

A1 - Coordinateur du Projet :

Un unique coordinateur doit être désigné par les partenaires.

Mme.	Christine PAULIN
Fonction[2]	Professeur
Laboratoire (Nom complet et sigle le cas échéant)[2]	Laboratoire de Recherche en Informatique LRI
Adresse[2]	LRI Bat 490, Université Paris Sud 91405 ORSAY Cedex
Téléphone[2]	01 69 15 66 35
Fax	01 69 15 65 86
Mél[2]	Christine.Paulin@lri.fr

[2]Champ obligatoire.

Action Concertée Incitative

SÉCURITÉ INFORMATIQUE

Descriptif complet du projet

A2- Équipes ou laboratoires partenaires du Projet [3] :

Identification de l'équipe ou du laboratoire

Équipe ou Laboratoire	Laboratoire d'informatique de l'Université de Franche-Comté, LIFC, FRE CNRS 2661
Adresse	16, route de Gray - 25030 BESANCON Cedex

Organisme de rattachement financier de l'équipe pour le présent projet

Université de Franche-Comté

Responsable du projet au sein de l'équipe ou du laboratoire

Mme.	Françoise BELLEGARDE[4]
Fonction	Responsable de l'équipe TFC (Techniques Formelles et à Contraintes) Responsable locale du projet INRIA CASSIS
Téléphone	03 81 66 64 52
Fax	03 81 66 64 50
Mél	bellegarde@lifc.univ-fcomte.fr

Autres membres de l'équipe participant au projet

Nom	Prénom	Poste statutaire	% du temps de recherche consacré au projet
Bouquet [4]	Fabrice	Maître de conférences	10 %
Giorgetti[4]	Alain	Maître de conférences	10 %
Julliand	Jacques	Professeur	10 %
Kouchnarenko[4]	Olga	Maître de Conférences	20 %
Legeard[4]	Bruno	Professeur	10 %
Masson	Pierre-Alain	Maître de Conférences	20 %
Mountassir	Hassan	Professeur	10 %

[3] Une fiche doit être remplie pour chaque laboratoire ou équipe partenaire.

[4] membre du projet CASSIS commun au LORIA et à l'université de Franche-Comté

Références :

Pour chaque (enseignant-)chercheur participant, liste de 3 à 5 publications, logiciels ou brevets les plus significatifs, en relation avec la thématique du projet.

- J. Julliand, P.A. Masson, H. Mountassir, *Modular verification for a class of PLTL properties*. International Workshop on Integrated Formal Methods, IFM'2000, Dagstuhl, Saarland, Germany, LNCS 1945, p. 398-419, Eds. B. Stodart, W. Grieskamp, T. Santen, Dagstuhl,2000.
- F. Bellegarde, C. Darlot, J. Julliand, O. Kouchnarenko, *Reformulation : a Way to Combine Dynamic Properties and B Refinement*. FME 2001 ("Formal Methods Europe"), LNCS 2021, Springer, p. 2-19, Berlin, Germany, 2001.
- F. Bellegarde, S. Chouali, J. Julliand. *Verification of Dynamic Constraints for B Event Systems under Fairness Assumptions*. 2nd International Conference of B and Z Users, ZB2002, Grenoble, France, LNCS 2272, p. 477-496, 2002.
- B. Legeard, F. Peureux, M. Utting. *A Comparison of the BTT and TTF Test-Generation Methods*. 2nd International Conference of B and Z Users, ZB2002, Grenoble, France, Proceedings LNCS 2272, p. 309-329, (janvier 2002).
- F. Bouquet, B. Legeard, F. Peureux. *CLPS-B - A Constraint Solver for B* In proc. of the conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'02, ETAPS, LNCS 2280, p.188-204, Springer, Grenoble, France, 2002.
- B. Legeard, *Automated Boundary-Value Test Generation from Specifications ? Method and Tools*. 4th International Conference on Software Testing, ICSTEST 2003, Cologne, Allemagne, (2-4 avril 2003), à paraître.
- C. Darlot, J. Julliand, O. Kouchnarenko. *Refinement Preserves PLTL Properties*. à paraître ZB'03, LNCS, Turku, Finland, 2003.
- J. Julliand, P-A. Masson, H. Mountassir. *Vérification par model-checking modulaire des propriétés dynamiques introduites en B*. Techniques et Science Informatique, Ed. Hermès, Vol. 20, 7, p. 927-957, 2001.
- A. Giorgietti, An asymptotic study for path reversal. A paraître dans la revue "Theoretical Computer Science".
- M. Tréhel, P. Gradit, A. Giorgetti, Performances d'un algorithme distribué d'exclusion mutuelle en cas de non-équiprobabilité des requêtes des processus. Numéro spécial "Evaluation quantitative des performances des réseaux et systèmes", Revue RSRCP (Réseaux et Systèmes Répartis, Calculateurs Parallèles), Ed. Hermès, Vol. 13, 6, p. 557-573, (2001).
- F. Bellegarde, C. Charlet, O. Kouchnarenko. *Raffiner pour vérifier des systèmes paramétrés*. Techniques et Sciences Informatiques, Editions Hermès, Vol. 8, 21, p. 1121-1149, 2002.
- B. Legeard, F. Peureux. *B-Testing-Tools : génération de tests aux limites à partir de spécifications B* Techniques et Sciences Informatiques, Hermès-Lavoisier, Vol. 21, 9, p. 1189-1218, 2002.
- F. Bouquet, B. Legeard, F. Peureux, and L. Py. Un système de résolution de contraintes ensemblistes pour l'évaluation de spécifications B. In *9ème Journées Francophones de Programmation Logique et Programmation par Contraintes, JFPLC'2000*, p 125–144, 2000.
- F. Bouquet and B. Legeard. Réification de scripts exécutables en génération de tests à partir de spécification formelles : application aux mécanismes de transaction de la java card. In *AFADL'2003*, p 141–156, 2003.
- F. Bellegarde and C. Charlet and O. Kouchnarenko, How to Compute the Refinement Relation for Parameterized Systems. Proc. First Int. ACM-IEEE Conf. on Formal Methods and Models for Codesign (MEMOCODE'2003), Mont St-Michel, 2003, à paraître.

Action Concertée Incitative

SÉCURITÉ INFORMATIQUE

Descriptif complet du projet

Identification de l'équipe ou du laboratoire

Équipe ou Laboratoire	LORIA Équipe CASSIS, INRIA Lorraine et Université de Franche-Comté [5]
Adresse	INRIA Lorraine 615 rue du jardin botanique BP 101 54602 VILLERS-LES-NANCY Cedex

Organisme de rattachement financier de l'équipe pour le présent projet

INRIA

Responsable du projet au sein de l'équipe ou du laboratoire

M.	Silvio RANISE
Fonction	Chargé de recherches INRIA
Téléphone	03 83 59 30 02
Fax	03 83 27 83 19
Mél	Silvio.Ranise@loria.fr

Autres membres de l'équipe participant au projet

Nom	Prénom	Poste statutaire	% du temps de recherche consacré au projet
Rusinowitch	Michael	Directeur de recherches	5 %
Deharbe	David	Maître de conférences (Brésil, membre associé)	20 %
Vigneron	Laurent	Maître de conférences	10%

[5] Les membres du projet CASSIS rattachés au Laboratoire d'Informatique de Franche-Comté apparaissent dans la description du LIFC

Références :

Pour chaque (enseignant-)chercheur participant, liste de 3 à 5 publications, logiciels ou brevets les plus significatifs, en relation avec la thématique du projet.

- A. Armando, S. Ranise, and M. Rusinowitch. A rewriting approach to satisfiability procedures. *Information and Computation*, to appear, 2003.
- A. Armando, S. Ranise, and M. Rusinowitch. Uniform derivation of decision procedures by superposition. In Laurent Fribourg, editor, *Proceedings of Annual Conference of the European Association for Computer Science Logic, CSL 01*, Lecture Notes in Computer Science, Cachan, September 2001. Springer Verlag.
- D. Déharbe and S. Ranise. *Bdd-driven first-order satisfiability procedures (extended version).* Research report 4630, INRIA, Nancy, France, November 2002.
- D. Déharbe and S. Ranise. haRVey : the two-face(s) of satisfiability checking. 2002–2003. `http://www.loria.fr/equipes/cassis/softwares/haRVey`.
- L. Bachmair, A. Tiwari, and L. Vigneron. *Abstract Congruence Closure.* Journal of Automated Reasoning, 2003 (To appear).
- M. Rusinowitch and L. Vigneron. *Automated Deduction with Associative-Commutative Operators.* Applicable Algebra in Engineering, Communication and Computation, 6(1) :23–56, January 1995.
- A. Bouhoula and M. Rusinowitch. *Observational Proofs by Rewriting.* Theoretical Computer Science, 275 (1–2) (2002) pp. 675–698.
- F. Klay, M. Rusinowitch, and S. Stratulat. *Mechanical Verification of an Ideal ABR Conformance Algorithm.* Journal of Automated Reasoning, 2003 (To Appear).
- A. Armando, M. Rusinowitch, and S. Stratulat. *Incorporating Decision Procedures in Implicit Induction.* Journal of Symbolic Computation, Vol. 34, No. 4, October 2001.

Action Concertée Incitative

SÉCURITÉ INFORMATIQUE

Descriptif complet du projet

Identification de l'équipe ou du laboratoire

Équipe ou Laboratoire	Projet LogiCal
Adresse	LRI UMR Université Paris Sud 91405 ORSAY Cedex INRIA FUTURS

Organisme de rattachement financier de l'équipe pour le présent projet

Université Paris Sud

Responsable du projet au sein de l'équipe ou du laboratoire

Mme.	Christine PAULIN
Fonction	Professeur, co-responsable du projet LogiCal
Téléphone	01 69 15 66 35
Fax	01 69 15 65 86
Mél	Christine.Paulin@lri.fr

Autres membres de l'équipe participant au projet

Nom	Prénom	Poste statutaire	% du temps de recherche consacré au projet
Contejean	Evelyne	Chargée de recherches CNRS	10 %
Filliâtre	Jean-Christophe	Chargé de recherches CNRS	30 %
Marché	Claude	Maître de Conférences UPS détaché à l'INRIA	30 %

Références :

Pour chaque (enseignant-)chercheur participant, liste de 3 à 5 publications, logiciels ou brevets les plus significatifs, en relation avec la thématique du projet.

- Y. Bertot, G. Dowek, C. Paulin-Mohring, and L. Théry, editors. *International Conference on Theorem Proving in Higher Order Logics (TPHOLs'99)*, Lecture Notes in Computer Science, Nice, September 1999. Springer-Verlag.
- E. Contejean, C. Marché, B. Monate, and X. Urbain. CiME version 2, 2000. Prerelease `http://cime.lri.fr/`.
- J.-C. Filliâtre. Verification of Non-Functional Programs using Interpretations in Type Theory. *Journal of Functional Programming*, February 2000. A paraître.
- J.-C. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS : Integrated Canonization and Solving (Tool presentation). In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings of CAV'2001*, volume 2102 of *Lecture Notes in Computer Science*, pages 246–249. Springer-Verlag, 2001.
- J.-C. Filliâtre. The Why certification tool. `http://why.lri.fr/`.
- C. Marché, C. Paulin, and X. Urbain. The Krakatoa tool for JML/Java program certification. Soumis `http://krakatoa.lri.fr`, 2003.
- C. Paulin-Mohring. Modelisation of timed automata in Coq. In N. Kobayashi and B. Pierce, editors, *Theoretical Aspects of Computer Software (TACS'2001)*, volume 2215 of *Lecture Notes in Computer Science*, pages 298–315. Springer-Verlag, 2001.
- The Coq Development Team. *The Coq Proof Assistant Reference Manual – Version V7.3*, May 2002. `http://coq.inria.fr`.
- The Krakatoa team. The Krakatoa proof tool, 2002. `http://www.lri.fr/~marche/krakatoa`.

Action Concertée Incitative

SÉCURITÉ INFORMATIQUE

Descriptif complet du projet

Identification de l'équipe ou du laboratoire

Équipe ou Laboratoire	Projet Lemme
Adresse	INRIA Sophia-Antipolis 2004, route des Lucioles BP 93 06902 Sophia-Antipolis

Organisme de rattachement financier de l'équipe pour le présent projet

INRIA Sophia-Antipolis

Responsable du projet au sein de l'équipe ou du laboratoire

Mme.	Marieke HUISMAN
Fonction	Chargée de Recherches INRIA
Téléphone	04 92 38 79 45
Fax	04 92 38 50 60
Mél	Marieke.Huisman@inria.fr

Autres membres de l'équipe participant au projet

Nom	Prénom	Poste statutaire	% du temps de recherche consacré au projet
Barthe	Gilles	Chargé de Recherches INRIA	10 %
Lanet	Jean-Louis	Ingénieur Spécialiste	10 %

Références :

Pour chaque (enseignant-)chercheur participant, liste de 3 à 5 publications, logiciels ou brevets les plus significatifs, en relation avec la thématique du projet.

- N. Cataño and M. Huisman. Chase : a Static Checker for JML's Assignable Clause. In L.D. Zuck, P.C. Attie, A. Cortesi, and S. Mukhopadhyay, editors, *Verification, Model Checking and Abstract Interpretation (VMCAI '03)*, LNCS 2575, p 26–40. Springer, 2003.
- N. Cataño and M. Huisman. Formal specification and static checking of Gemplus's electronic purse using ESC/Java. In L.-H. Eriksson and P.A. Lindsay, editors, *Formal Methods Europe (FME'02)*, LNCS 2391, p 272–289. Springer, 2002.
- K. Trentelman and M. Huisman. Extending JML Specifications with Temporal Logic. In H. Kirchner and C. Ringeissen, editors, *Algebraic Methodology And Software Technology (AMAST'02)*, LNCS 2422, p 334–348. Springer, 2002.
- G. Barthe, D. Gurov, and M. Huisman. Compositional verification of secure applet interactions. In R.-D. Kutsche and H. Weber, editors, *Fundamental Approaches to Software Engineering (FASE'02)*, LNCS 2306, p 15–32. Springer, 2002.
- G. Barthe and P. Courtieu. Efficient reasoning about executable specifications in Coq. In V. Carreño, C. Muñoz, and S. Tahar, editors, *Theorem Proving in Higher Order Logics (TPHOLs'02)*, LNCS 2410, p 31–46. Springer, 2002.
- G. Barthe, P. Courtieu, G. Dufay, and S. Melo de Sousa. Tool-Assisted Specification and Verification of the JavaCard Platform. In H. Kirchner and C. Ringeissen, editors, *Algebraic Methodology And Software Technology (AMAST '02)*, LNCS 2422, p 41–59. Springer, 2002.
- L. Burdy, A. Requet, and J.-L. Lanet. Java Applet Correctness : a Developer-Oriented Approach, 2003. Manuscript.
- P. Bieber, J. Cazin, P. Girard, J.-L. Lanet, V. Wiels, and G. Zanon. Checking secure interactions of smart card applets. *Journal of Computer Security*, 10(4) :369–398, 2002.
- J.-L. Lanet. Are smart cards the ideal domain for applying formal methods ? In J.P. Bowen, S. Dunne, A. Galloway, and S. King, editors, *ZB 2000 : Formal Specification and Development in Z and B*, LNCS 1878, p 363 – 374. Springer, 2000.

Action Concertée Incitative

SÉCURITÉ INFORMATIQUE

Descriptif complet du projet

Identification de l'équipe ou du laboratoire

Équipe ou Laboratoire	LSR - IMAG, équipe VaSCo
Adresse	Laboratoire Logiciels, Systèmes, Réseaux Domaine Universitaire B.P. 72 38 402 Saint Martin d'Hères cedex

Organisme de rattachement financier de l'équipe pour le présent projet

Institut National Polytechnique de Grenoble

Responsable du projet au sein de l'équipe ou du laboratoire

Mme.	Marie-Laure Potet
Fonction	Maître de Conférences Habilité
Téléphone	04 76 82 72 69
Fax	04 76 82 72 87
Mél	Marie-Laure.Potet@imag.fr

Autres membres de l'équipe participant au projet

Nom	Prénom	Poste statutaire	% du temps de recherche consacré au projet
Bert	Didier	Chargé de Recherche CNRS	10 %
Boulmé	Sylvain	Maître de Conférences	20 %
Oriat	Catherine	Maître de Conférences	10 %

Références :

Pour chaque (enseignant-)chercheur participant, liste de 3 à 5 publications, logiciels ou brevets les plus significatifs, en relation avec la thématique du projet.

- D. Bert, M-L. Potet, Y. Rouzaud. *A study on components and assembly primitives in B* Proc. of First B Conference, IRIN, Nantes, novembre 1996
- D. Bert, F. Cave. *Construction of Finite Labelled Transition Systems from B Abstract Systems* IFM 2000, LNCS 1945, Springer-Verlag, novembre 2000
- H. Ruiz Barradas, D. Bert. *Specification and Proof of Liveness Properties under Fairness Assumptions in B Event Systems* IFM 2002, LNCS 2335, springer-Verlag, Mai 2002
- F. Badeau, D. Bert, S. Boulmé, M-L. Potet, N. Stouls, L. Voisin. *Traduction de B vers des langages de programmation : points de vue du projet BOM* Actes des Journées AFADL, IRISA, RENNES, Janvier 2003
- S. Boulmé. *Spécification d'un environnement dédié à la programmation certifiée de bibliothèques de calcul formel* Mémoire de thèse, Université Paris 6, décembre 2000
- S. Boulmé *Opérateurs de raffinement sur les structures algébriques* Journées Francophones des Langages Applicatifs, INRIA, février 2000
- S. Boulmé. *Toward a High Order Functional and Imperative Specification Language* Rapport interne, février 2003
- C. Oriat. *Detecting equivalence of modular specifications with categorical diagrams.* Theoretical Computer Science, vol 247, 2000
- Y. Ledru, C. Oriat, M.-L. Potet. *Le raffinement vu comme primitive de spécification — une comparaison de VDM, B et Specware* Approches formelles dans l'assistance au développement de logiciel AFADL'98, Poitiers, 1998
- O. Maury, C. Oriat, Y. Ledru. *Invariants de liaison pour la cohérence de vues statique et dynamique en UML.* AFADL'01, Nancy, 2001
- M.-L. Potet, Y. Rouzaud. *Composition and Refinement in the B-Method* Second International B Conference, LNCS 1393, Springer-Verlag, 1998
- P. Bontron, M.-L. Potet *Automatic Construction of Validated B components from Structured Developments* ZB 2000, LNCS 1878, Springer-Verlag, 2000
- M-L. Potet. *Spécifications et développements formels : Etude des aspects compositionnels dans la méthode B.* Habilitation à Diriger des Recherches, INPG, décembre 2002
- M-L. Potet. *Spécifications et développements structurés dans la méthode B.* Technique et Science Informatiques, vol 22, février 2003

Action Concertée Incitative

SÉCURITÉ INFORMATIQUE

Descriptif complet du projet

B - DESCRIPTION DU PROJET

B1 – Objectifs et contexte :

On précisera, en particulier, les verrous scientifiques et technologiques à dépasser, l'état de l'art ainsi que les projets concurrents ou similaires connus dans le contexte national et international, en particulier ceux auxquels les équipes du projet participent.

État de l'art L'accroissement du nombre d'applications informatiques critiques dans les domaines des transports, du commerce, des télécommunications ou du commerce électronique a rendu nécessaire de développer des méthodes et outils permettant de suivre la conception des systèmes de la phase de spécifications jusqu'à la génération de code. La rigueur de ce développement est un élément important dans le processus de certification des applications. L'utilisation de modèles formels est par exemple nécessaire pour obtenir les degrés les plus hauts de certification suivant les critères communs.

Un exemple de méthodologie de développement est donné par la méthode B de J.-R. Abrial [1]. Cette méthode est mise en œuvre dans les outils commerciaux Atelier B [3] ou B-Tools [4] ; l'un de ses succès a été la formalisation du système de freinage d'urgence du métro METEOR. Cette méthode repose sur la notion de *machines abstraites* qui encapsulent un état et des opérations de transformation de cet état. Les machines sont développées par raffinements successifs jusqu'à obtenir une description assez concrète pour pouvoir être traduite en code exécutable. Un générateur d'obligations de preuve calcule des conditions qui garantissent la validité du raffinement. Un assistant de preuves assiste ensuite l'utilisateur pour résoudre ces conditions. Ces machines sont exprimées (spécification et code) dans un langage ad-hoc mélangeant notations mathématiques et quelques instructions de code (affectations, conditionnelles...).

Un autre point de vue pour spécifier et vérifier du code consiste à travailler directement au niveau du code source en insérant des annotations logiques (invariants, pré et post-conditions). Un exemple de cette approche est le langage JML (Java Modeling Language) [32, 28] qui permet de spécifier du code Java. Le langage de JML repose sur un sous-ensemble de Java sans effet de bord qu'il étend par des constructions spécifiques (quantification, connecteurs logiques, accès au résultat d'une méthode, aux anciennes valeurs des variables ...). Les spécifications sont incluses comme commentaires Java spécifiques. L'idée est que ce langage et les annotations s'insèrent naturellement dans le processus de développement des ingénieurs. En effet la sémantique de JML étend de manière naturelle celle de Java et ne nécessite pas l'apprentissage pour les ingénieurs d'un nouveau formalisme ou d'une méthode spécifique de développement. D'autre part plusieurs outils existent déjà avec des objectifs variés pour traiter des programmes Java annotés par des spécifications JML. On pourra trouver une description synthétique de ces outils et des références dans [17].

Il est possible d'interpréter les annotations JML qui sont de simples commentaires Java de différentes manières. On peut simplement les ignorer, ou les utiliser comme commentaires (un outil jmldoc permet de construire une documentation hyper-texte) ; on peut utiliser les annotations comme oracle pour construire des moteurs de test (c'est ce qui est fait par jmlunit) on peut chercher par une analyse statique automatique si le code viole une des annotations (ainsi l'outil ESC/Java développé par Compaq [34] détecte l'accès à des champ d'un objet null ou en dehors des bornes d'un tableau, l'outil Chase [22] traite les conditions de *frame* qui spécifient les références non modifiées, ces deux outils reposent sur des méthodes qui ne sont ni sûres ni complètes) ; on peut utiliser les annotations pour engendrer un code défensif qui échouera si les méthodes sont utilisées de manière non conforme à leur spécification, c'est ce qui est fait dans un outil comme jmlc ou bien Jass [8] qui utilise un sous-ensemble de JML mais permet de plus de spécifier des traces d'exécution, (cette approche nécessite que les propriétés à vérifier correspondent à des codes exécutables) ; finalement on peut utiliser les annotations d'invariant, de pré et post-conditions pour engendrer de manière sûre des obligations de preuves suffisantes pour garantir la correction des programmes (LOOP [35, 12, 40], JIVE [31, 38], Krakatoa [30, 36], Jack [18, 19, 20]). Ces outils utilisent un modèle de la sémantique des programmes objets exprimé en théorie des ensembles ou en logique d'ordre supérieure. Des obligations de preuve sont engendrées qui peuvent être résolus à l'aide d'un assistant de preuve (PVS, HOL, Coq ou le prouveur de l'atelier B).

L'approche Java/JML a reçu un accueil très favorable dans le domaine de la vérification des applets pour cartes à puces ou terminaux utilisant la technologie Java [15, 21]. Cependant l'annotation et la vérification a posteriori des programmes reste une tâche demandant une expertise importante. Des travaux existent (mis en œuvre dans l'outil Daikon développé au MIT) qui automatisent la recherche d'invariants de classes par généralisation de propriétés à partir d'un ensemble de traces d'exécution.

Les propriétés à montrer pour garantir la correction de programmes requièrent de raisonner sur des domaines infinis (arithmétique, structures de données). Elles sont en général en dehors des fragments de logique traités par les démonstrateurs automatiques. Deux techniques différentes peuvent être mises en œuvre pour remédier à ce problème. La première consiste à chercher à établir des propriétés simples de programme, à calculer les conditions que le programme doit satisfaire pour vérifier ces propriétés sous la forme d'un ensemble de formules du premier ordre qui sont ensuite traitées par un démonstrateur approprié. Le système ESC/Java fonctionne sur ce modèle. Il s'appuie sur l'outil de preuve Simplify qui combine raisonnement propositionnel, procédures de vérification de satisfiabilité (pour un fragment de l'arithmétique, une théorie de l'égalité sans quantificateur et la théorie des tableaux) et des heuristiques pour deviner les variables d'instanciations. Si une formule ne peut être prouvée, Simplify essaie de deviner un contre-exemple à partir de l'ensemble de littéraux. Le caractère heuristique de la procédure fait que certaines formules valides ne pourront être démontrées et que les contre-exemples fournis peuvent être erronés ou incomplets. La seconde approche consiste à abstraire le programme pour en trouver un modèle fini préservant le flot de contrôle du programme initial. Cette abstraction utilise des variables booléennes pour modéliser les prédicats intervenant dans le programme. La phase de vérification cherche à établir par des techniques de model-checking que l'on ne peut atteindre d'état dans lequel la propriété à satisfaire est invalidée. Si c'est le cas, le programme initial est correct, sinon il faut raffiner l'abstraction en utilisant des démonstrateurs tels que Simplify pour éliminer certains chemins non réalisables et raffiner l'abstraction. Ce paradigme "abstraire, vérifier, raffiner" est utilisé dans un système comme SLAM [7] pour prouver des propriétés de programme C.

Dans ces deux approches à la vérification de programme, il est essentiel de disposer d'outils de preuve qui soient à la fois puissants, automatiques et flexibles.

De nombreux outils (CVC, Simplify, SVC et STeP) traitent le problème de la validité de fragments de la logique du premier ordre avec égalité et modulo certaines théories avec différents degrés de succès. Ils ont montré leur intérêt dans certains contextes de vérification, mais souffrent de deux limitations majeures : la gestion des quantificateurs et une sensibilité extrême à tout changement de théorie. Le système Simplify apporte une réponse partielle au problème des quantificateurs, par des heuristiques d'instanciation des variables quantifiées fondées sur le filtrage. L'outil haRVey développé dans l'équipe CASSIS combine du raisonnement propositionnel à base de BDD (Binary Decision Diagram) et équationnel par superposition, afin de vérifier la validité de formules complexes, avec quantificateurs et modulo diverses théories. Cet outil privilégie l'automatisation de la preuve, qui doit être transparente pour l'utilisateur final, en cernant finement sa portée. Les premiers résultats positifs ont abordé le débogage de programmes séquentiels.

Verrous scientifiques et technologiques L'approche Java/JML est intéressante car elle est facilement accessible aux ingénieurs. Elle dispose déjà d'une palette intéressante d'outils offrant des degrés de vérification du programme variés (documentation, test, analyse statique, preuve) pouvant être utilisés de manière automatique ou assistée. JML permet simplement de modéliser des versions légères de spécification : quelles exceptions sont levées ou quelles cellules ne sont pas modifiées.

Cependant, on sait depuis longtemps que l'annotation et la vérification a posteriori d'un code sont des tâches très laborieuses. Les preuves sont d'autant plus complexes que la spécification s'exprime en termes abstraits éloignés du code. Les spécifications JML apparaissent trop souvent comme une paraphrase du code. Pourtant, dans le processus de certification de logiciel critique, il est essentiel d'établir le lien entre le code exécuté et une propriété de haut niveau telle que : certaines erreurs systèmes ne peuvent se produire, ou bien l'accès à telle donnée ne peut se faire qu'après identification.

Les techniques de raffinement qui sont à la base de la méthode B peuvent apporter des solutions à ce problème. De même la notion de `model` de JML offre des pistes à une spécification qui manipule des objets abstraits. Cependant le lien entre le raffinement entre classes et le raffinement des machines abstraites de B reste à établir. Il convient également d'étudier sous quelles conditions le raffinement va permettre de garantir la préservation des propriétés de sécurité que l'on cherche à établir.

La programmation objet favorise le développement et la réutilisation de bibliothèques génériques dans des contextes diversifiés. Savoir spécifier de manière abstraite ces opérateurs est un challenge. La sémantique des langages objets est complexe (héritage, liaison tardive, règles de visibilité, initialisation ...). Cette complexité se retrouve au niveau du langage JML qui fait intervenir des expressions Java supposées sans effet de bord. Un problème non résolu actuellement est de trouver les bonnes restrictions sur la spécification d'une méthode redéfinie dans une sous-classe pour garantir une bonne réutilisation des preuves déjà établies.

Les méthodes automatiques de vérification de ESC/Java ou Chase ne sont actuellement ni correctes ni complètes, ce qui est un compromis pragmatique mais peu satisfaisant. Pourtant le déploiement des techniques de preuve ou de raffinement de programmes dépend fortement de leur automatisation. Les obligations de preuve engendrées automatiquement tendent à grossir, le fait qu'elles portent sur le modèle mathématique des programmes objets nuit également à leur lisibilité. Il est donc nécessaire que le maximum de ces obligations puissent être prouvées automatiquement, laissant l'utilisateur avec celles nécessitant un raisonnement intelligent ou celles non prouvables qui identifient une erreur de spécification ou de programme. Il faut donc proposer des outils de preuve pour attaquer les obligations de preuve engendrées. Pour cela, il faut identifier la nature des obligations de preuve engendrées et mettre en place des outils efficaces et corrects pour les résoudre.

D'autre part, dans le processus de développement du code, une grande partie du temps n'est pas consacrée à prouver la correction du développement mais plutôt à identifier les erreurs et tenter de les réparer. Un environnement de développement doit donc faciliter cette tâche en permettant en particulier de simuler le comportement du programme dans les cas où les obligations de preuve ne sont pas prouvables, où de vérifier le comportement du programme dans des cas particuliers.

La garantie du logiciel s'appuie essentiellement sur une modélisation rigoureuse (du programme et des propriétés qu'il doit vérifier) et l'établissement de preuve de correction. Cependant une telle approche est, à juste titre, réputée trop abstraite et nécessite une expertise incompatible avec sa généralisation dans le processus de développement de logiciel sécurisé à l'échelle industrielle. Cependant les techniques de preuve peuvent s'intégrer plus naturellement dans le procédé de développement de logiciel : le langage de spécification doit se rapprocher du domaine de compétence de celui qui le manipule (sécurité, programmation). Le développement doit être modulaire et réutilisable, tant au niveau du code que de celui des spécifications. Les informations sur les obligations de preuve sont également utiles pour favoriser la localisation d'erreurs, engendrer des tests ou produire un code sûr par une approche défensive.

Autres projets Les membres des projets Lemme et LogiCal sont impliqués dans le projet IST VERIFICARD[6] qui se termine en septembre 2003. Dans le cadre de ce projet, ont été étudiés la formalisation de la plate-forme JavaCard (vérificateur, compilateur,...) ainsi que des méthodes et outils de spécification modélisation et vérification d'applets. Ce projet a promu l'utilisation du langage JML pour les spécifications d'applets pour cartes à puce et soutenu le développement de LOOP, Jive, Krakatoa et Chase. Le travail proposé dans le cadre de cette ACI s'appuie sur l'expérience acquise dans le projet VERIFICARD et s'attaquera à différents problèmes de recherche mis en évidence lors de ce projet.

Les membres des équipes TFC et VasCo ont participé au projet RNTL INKA (novembre 2000-décembre 2002) qui se poursuit dans le projet DANOCOPS labellisé en 2002. L'objectif de ce projet est la réalisation d'un outil pour détecter des non-conformités aux spécifications en utilisant les techniques de programmation par contraintes. L'outil développé dans le cadre du projet INKA génère automatiquement et de manière déterministe des données d'entrées à partir de la caractérisation des fragments de code à atteindre. Il utilise une technique originale s'appuyant sur la résolution de systèmes de contraintes. L'objectif du projet DANOCOPS est d'exploiter les potentialités de cette technique pour la recherche de défauts de conformité. Il s'agit de développer un outil capable d'appliquer cette méthode sur des programmes écrits en C/C++/JAVA et des spécifications écrites en UML - OCL ainsi que d'explorer l'application de cette technique dans le cas de spécifications décrites en B et Lustre.

Les équipes VasCo et TFC participaient au projet RNTL précompétitif B-OM (avril 2001-mars 2003) dont l'objectif était le développement d'un traducteur B optimisant la mémoire, afin de répondre aux contraintes carte à puce. Le développement par raffinement d'un interpréteur de Byte code a permis de valider l'approche formelle partant d'une spécification de haut niveau jusqu'à du code embarquable.

Le projet GECCOO s'inscrit dans le Réseau Thématique Pluridisciplinaire n°19 - SECC Systèmes Embarqués Complexes ou Contraints[7] auquel participent les équipes et projet Cassis, LogiCal, TFC et Vasco. Le projet Lemme participe à l'action spéciale Sécurité au sein de ce réseau.

B2 – Description du projet : (5 à 10 pages)

Entre autres, le caractère innovant du projet (concepts, technologies, expériences ...) devra être explicité et la valeur ajoutée des coopérations entre les différentes équipes sera discutée.

Notre projet se propose d'élaborer des méthodes et outils pour développer dans des formalismes orientés objet des applications pour lesquelles les exigences de correction sont élevées. L'originalité du projet est de chercher à construire une chaîne cohérente de développement, de la phase de spécification jusqu'à la

[6] http://www.verificard.org

[7] http://www.systemes-critiques.org/SECC/

génération de code sûr en passant par les étapes de raffinement, de simulation et de test. La notion de preuve est au centre des techniques que nous mettons en œuvre. Chaque étape du développement doit être justifiée formellement et si possible de manière mécanisée. Toutes les propriétés à montrer pour assurer la correction d'une application logicielle ne peuvent en général pas être montrées automatiquement pour des raisons théoriques d'indécidabilité des logiques concernées. Les prouver pas à pas de manière interactive est théoriquement faisable mais en pratique trop lourd à mettre en œuvre. Il est donc important d'une part de développer des méthodes automatiques adaptées aux classes de problèmes rencontrés dans les preuves de propriété de sécurité des logiciels, d'autre part d'utiliser au mieux la combinaison d'approches automatiques et interactive (en particulier tests dynamiques ou statiques et preuves) pour produire in fine un code correct.

Les équipes qui collaborent dans ce projet ont des compétences complémentaires dans les domaines de la sécurité, du raffinement, des modèles de programmation objet, de la démonstration interactive et automatique et de la preuve de programme. Des collaborations ont déjà été établies, ainsi l'outil de preuve de programmes Why [26, 25] a été adapté par J.-C. Filliâtre du projet LogiCal pour pouvoir engendrer des obligations de preuve qui peuvent être résolues par haRVey développé par S. Ranise au LORIA. Les équipes du LIFC et du LORIA collaborent au sein du projet commun CASSIS. Elles ont également collaboré avec l'équipe VasCo du LSR dans le cadre d'un projet RNTL B-OM visant à optimiser le code engendré par la méthode B en vue de l'embarquer sur des cartes à puce. Les projets Lemme et LogiCal ont collaboré en particulier au sein du projet IST Européen VERIFICARD dans lequel se sont développées entre autres les approches JML/JavaCard pour la vérification du code des applets.

Le projet s'articulera autour de quatre thèmes principaux qui correspondent aux différentes étapes de développement d'un programme devant offrir des garanties de sécurité.

1. Spécifications de haut niveau
2. Raffinement et programmation orientée objet
3. Génération et résolution d'obligations de preuve
4. Détection d'erreurs

À ces quatre thèmes, s'ajoutera un thème transversal qui consistera à mettre en place une *chaîne de développement* de programmes sécurisés.

Tout d'abord, il s'agit d'identifier un *langage de spécification* de haut niveau permettant d'exprimer des politiques de sécurité couramment utilisées et pouvant se traduire en terme d'invariants et de pré et post-conditions sur des classes et des méthodes.

Une problème important est celui de fournir un langage de spécification adapté à un *développement par raffinement*. Le raffinement devra permettre de garantir la préservation des propriétés de sécurité, il devra également être intégré au modèle de développement objet par héritage et redéfinition.

La correction des étapes de raffinement et finalement du code s'appuie sur *la génération et la résolution d'obligations de preuve*. La complexité des propriétés à montrer dépend de la théorie dans laquelle elles sont exprimées et des exigences sur ce que l'on souhaite établir. Par exemple les outils actuels ne garantissent que la correction partielle des méthodes récursives. On peut ne s'intéresser qu'à la propagation d'erreurs ou à la non modification de certaines données. Il faut mettre en relation les propriétés à montrer et les techniques de preuves afin que le modèle permette de traiter le plus grand nombre possible d'obligations de manière automatique.

L'annotation d'un code par des assertions a d'autres applications que la vérification formelle en particulier la *détection d'erreurs*. En effet ces annotations peuvent permettre d'identifier par résolution de contraintes des jeux de tests permettant d'atteindre un point de programme particulier. Ils peuvent servir à produire un code défensif qui interdira l'exécution de code en dehors des conditions spécifiées. C'est également un outil de simulation qui peut identifier de manière précoce des erreurs dans le code ou la spécification.

Les équipes impliquées développent des outils permettant de mettre en œuvre et de valider les solutions proposées. L'objet de cette ACI n'est pas le développement d'une plate-forme unique qui nous semble prématurée à ce stade d'avancement. Cependant, il est important de veiller à la cohérence de la *chaîne de développement* et de proposer des prototypes intégrant, au moins partiellement, les techniques proposées. Cela sera réalisé en étendant, en combinant ou en adaptant les outils actuellement développés par les partenaires du projet. Ceci facilitera en particulier l'évaluation des approches proposées sur des exemples concrets. Nous avons des contacts avec des organismes (Gemplus, NASA, Trusted Logic) qui ont montré leur intérêt dans les approches que nous développons et nous ont proposé des cas d'étude intéressants.

B2-1 Spécifications de haut niveau

Participants Lemme,VasCo

Cette activité combine des aspects méthodologiques (quelles sont les propriétés que nous voulons exprimer) et des aspects sémantiques (que signifie cette propriété). Le point de départ est le langage JML tel qu'il est défini actuellement. Nous envisageons d'étudier l'expression des propriétés qui nous intéressent en JML et de proposer des extensions si nécessaire. Nous nous concentrons sur les applications dans le domaine de la carte à puce. Plus spécifiquement les points suivants seront abordés :

- Définition d'un langage formel pour exprimer les propriétés de sécurité ;
- Définition précise d'une notion d'invariant de classe adaptée au modèle orienté objet ;
- Développement d'une méthode de construction de spécification par assemblage.

Langage formel pour exprimer les propriétés de sécurité Le langage JML tel qu'il est actuellement permet d'exprimer par exemple des invariants de classe, des pré et post conditions. Ces constructions de spécification restreignent le comportement d'une méthode particulière ; elles peuvent être vérifiées méthode par méthode, en utilisant des techniques classiques de logique de Hoare et de calcul de plus faible précondition. Cependant, dans ce projet, nous nous intéressons à des propriétés de sécurité telles que :

- contrôle d'accès : une commande ne peut être exécutée sans être en mode authentifié ;
- intégrité : une donnée partagée ne peut être modifiée qu'en mode authentifié par un tiers de confiance ;
- confidentialité : un donnée protégée ne peut être accédée qu'en mode authentifié par un tiers de confiance ;
- disponibilité du service : une demande est toujours suivie d'une réponse ;
- sûreté : une certaine classe d'exceptions n'est jamais levée ;
- enchaînement de méthodes : les méthodes sont seulement appelées dans un certain ordre ou un nombre limité de fois etc.

Beaucoup de ces propriétés ne peuvent pas s'exprimer directement au niveau des méthodes. C'est pourquoi nous avons besoin d'un niveau plus élevé de spécification qui peut être utilisé pour restreindre l'interaction entre différentes méthodes (et objets).

Dans la littérature, des méthodes différentes existent pour exprimer ces propriétés (pré-conditions, automates qui décrivent les enchaînements autorisés des traitements, etc.). Nous nous proposons de les étudier dans un premier temps puis d'intégrer la méthode la plus adaptée à JML, en gardant à l'esprit l'un des objectifs dans le design de JML : une spécification JML doit être facilement compréhensible par un programmeur Java ordinaire.

Pour définir la sémantique de ce langage nous commençons par définir un procédé d'insertion d'assertions JML dans les applications Java. L'idée est de synthétiser, à partir d'une application et d'une politique de sécurité, une application annotée ; la contrainte est ici d'assurer que les annotations garantissent bien que l'application adhère à la politique de sécurité. Pour les politiques les plus simples, il est vraisemblable que le langage JML soit approprié, mais les propriétés de sécurité les plus complexes pourraient nécessiter le recours à des extensions de JML. La sémantique pour cette extension reposera sur les principes d'encapsulation et le modèle objet développés dans la sous-tâche suivante.

Le travail final consistant à engendrer une application sécurisée à partir de l'application annotée en JML est décrit dans la tâche 4.

Le principe d'encapsulation Afin de définir la sémantique des extensions proposées, nous avons besoin de définir un modèle objet avec une granularité appropriée. Du fait des propriétés que nous cherchons à vérifier, le modèle objet doit nous permettre de parler du comportement des objets au niveau des appels de méthode. Pour cela nous avons besoin de définir formellement le principe d'encapsulation pour les langages Java et JML prenant en compte la composition (relation *has-a*) et l'héritage (relation *is-a*). Le modèle objet doit coïncider avec les sémantiques pour Java et JML comme elles ont déjà été définies, par exemple dans le projet LOOP [35].

En particulier, la notion d'invariant, qui est essentielle dans les spécifications objet, parce qu'elle permet d'exprimer des propriétés sur les attributs des objets qui doivent être vérifiées durant tout le cycle de vie de ces objets, doit garder son sens usuel : une propriété valide dans tous les états observables des objets sur lesquels elle porte [1].

Un point de départ pour définir un modèle objet approprié est de représenter les objets comme des co-algèbres (plus grands point fixes) [27]. Cela donne immédiatement une définition pour les invariants et les bi-simulations (équivalence de comportements). Cependant, à cause de la possibilité d'avoir différents modificateurs d'accès en Java, et parce que Java n'encapsule pas toujours complètement ses objets, nous

devrons adapter finement le modèle objet afin de capturer fidèlement la sémantique de Java. Un point particulièrement intéressant est de définir à quelles conditions les invariants et bi-simulations sont préservées par composition et héritage.

Cette étude pourra amener à faire des choix sur la portée et l'expressivité des invariants, ainsi que sur la structuration des applications (maîtrise des alias et du partage par exemple). Ces choix devront être validés du point de vue méthodologique. La vérification de la notion d'invariant, et les propriétés compositionnelles possibles, seront étudiés dans la tâche 3.

La construction de spécification par assemblage Un des aspects cruciaux permettant de maîtriser la complexité des applications est la construction de spécifications par assemblage. Il est d'ailleurs à la base du développement objet qui permet de réutiliser et d'étendre des classes, suivant les besoins. Dans le cadre du développement formel, il doit donc être possible de construire de manière incrémentale des spécifications, en réutilisant à la fois le code, les spécifications et les vérifications. Cet aspect n'est pas supporté par le langage d'assertions JML, qui ne permet d'énoncer des assertions que sous la forme explicite de relations avant-après. Nous nous proposons dans ce projet d'étudier un mode de spécification manipulant explicitement des transformateurs d'états, comme ceci est proposé dans le calcul du raffinement [6], et adapté au développement objet. Il deviendra alors possible de construire de nouvelles spécifications en composant par exemple des appels de méthodes, sans avoir besoin d'exprimer la spécification résultante sous forme de relation avant-après. En dehors de la facilité de spécification, de telles constructions peuvent aussi offrir des garanties directes en terme de vérification compositionnelle, si les opérateurs de composition préservent les propriétés attendues. De plus, les spécifications construites en combinant des transformateurs d'états permettent d'uniformiser les spécifications et le code et d'introduire un aspect opérationnel dans les spécifications. Ce dernier point est important dans le cadre des approches objets. Il permet d'une part d'expliciter les interactions possibles entre composants et d'autre part de spécifier des traitements reposant sur la liaison tardive [16].

Les axes développés seront les suivants :
- étude des besoins pour la spécification d'applications objet (instances, liaison tardive, etc.) ;
- choix de l'intégration Java-JML pour exprimer des spécifications ;
- sémantique de ce langage par calcul de plus faible précondition ;
- étude des propriétés de compositionnalité des invariants pour ce langage.

B2-2 Raffinement et programmation orientée objet

Participants VasCo,TFC,Lemme

La seconde tâche du projet GECCOO s'intéresse au développement formel des spécifications de haut niveau, telles que définies dans la tâche 1 et de programmes Java card, en incluant la préservation des propriétés de sécurité exigées. Cette tâche a donc pour but d'étendre et d'intégrer les travaux déjà réalisés par les partenaires sur la preuve de programmes objets et sur le raffinement modulaire dans un cadre non-objet. En effet, les environnements de preuve de programmes objet n'incluent actuellement ni la redéfinition ni le raffinement de données. De la même manière les travaux développés dans le cadre de la méthode B n'incluent pas l'héritage et la structuration objet, en particulier la création d'instances n'est pas explicitement traitée. Cette tâche s'appuiera sur l'expérience des partenaires de la mise en pratique d'applications structurées par raffinement.

Cette tâche contient à la fois des aspects méthodologiques (comment construire et représenter les niveaux de raffinement) et des résultats techniques sur le raffinement du langage de spécification et la préservation des propriétés de sécurité par raffinement. Cette tâche sera menée de manière incrémentale, en levant certaines restrictions introduites sur les langages (prise en compte de certaines formes d'alias) et la structuration des applications (prise en compte de certaines formes de partage). Plus précisément nous serons amenés à :
- définir un calcul du raffinement pour le langage de spécification issu de la tâche 1 et pour le sous-ensemble de Java considéré dans les implémentations ;
- définir les propriétés du raffinement en lien avec la structuration objet : héritage, composition sans partage, composition avec partage et prise en compte des interactions entre objets ;
- étudier le raffinement en lien avec les propriétés de sécurité introduites dans la tâche 1, de manière à garantir ces propriétés.

L'objectif final de cette tâche sera de produire le cadre formel permettant la construction des obligations de preuve assurant la correction du raffinement et la préservation des propriétés de sécurité, ces obligations de preuve constituant l'entrée de la tâche suivante.

Raffinement objet La première sous-tâche devra préciser une notion de raffinement au niveau des classes et des méthodes, en cohérence avec la notion de composant telle que définie dans la tâche 1 (principe d'encapsulation et portée des invariants). Ce travail sera mené en étendant aux composants objet le raffinement proposé dans la méthode B [1], qui permet de raffiner des données encapsulées. Dans cette approche le principe de substitutivité (remplacement d'un composant par son raffinement) est transparent à l'utilisateur, en particulier parce que les interfaces (assimilées dans ce cadre aux profils des opérations) ne peuvent pas être raffinées. De la même manière JML offre une certaine notion de raffinement [32], à travers les variables de modèle et les raffinements sont représentés par une relation d'héritage. Une première étude consistera à préciser le lien entre ces deux formes de raffinement [16] et la manière de les représenter. Un aspect sémantique important sera de définir le principe de substitutivité [5] associé à la notion de raffinement. La solution retenue dans JML, d'assimiler le raffinement à une certaine forme d'héritage, n'est pas nécessairement pleinement satisfaisante. En effet l'héritage (utilisé ici comme une relation de sous-typage) permet potentiellement de renforcer les préconditions, ce qui n'est pas le point de vue du raffinement. D'autre part, le principe de substitution associé à l'héritage du langage Java (sous-typage appliqué uniquement à l'objet sur lequel s'applique les méthodes) introduit un principe de substitution limité, qui ne permet pas complètement de raisonner sur des modèles abstraits, et donc de construire des preuves plus simples.

Structuration et raffinement La sous-tâche précédente a pour but de définir le raffinement au niveau des composants. Le but ici est d'étendre ce raffinement au niveau des applications structurées, en incluant la construction de spécifications par assemblage, telle que décrite dans la tâche 1. L'objectif visé sera d'étudier des conditions permettant d'assurer la compositionnalité des preuves [43], afin de pouvoir raffiner de manière indépendante des classes et de composer ces raffinements. Les axes développés seront les suivants :

- étude des propriétés des opérateurs du langage d'assertions vis à vis du raffinement (pour une relation de raffinement donnée) ;
- étude des structurations objets (héritage, composition et interactions entre objets) en lien avec le principe de substitutivité du raffinement. Ceci nécessitera en particulier d'expliciter la composition des relations de raffinement de données ;
- analyses des formes de structuration permettant d'assurer la compositionnalité des preuves.

Dans cette sous-tâche, le but sera d'étendre au cadre objet l'approche retenue dans la méthode B, pour laquelle nous avons déjà défini un cadre théorique [42, 41]. Basé sur le principe de substitutivité associé aux composants, ce cadre décrit la composition des relations de raffinement en fonction des différentes primitives de structuration du langage et énonce des règles architecturales permettant de maîtriser le partage entre objets et la non-introduction d'alias, au cours du raffinement des données. Les extensions proposées devront prendre en compte les mécanismes de structuration objet et en particulier l'héritage qui, d'un point de vue méthodologique, correspond à différentes formes d'utilisation : définitions par défaut, raffinement ou sous-typage. Une tâche méthodologique consistera à éclaircir ces points et à offrir des mécanismes permettant de raisonner sur les structurations horizontale et verticale, afin de déduire des propriétés globales sur les applications structurées.

Préservation des propriétés de sécurité Le raffinement tel que décrit ci-dessus est lié à la granularité des méthodes et assure l'observabilité des valeurs des états, à la relation de représentation près. Ce type de raffinement préserve donc, par construction, les propriétés de sûreté sur les états. Les propriétés de sécurité visées dans le projet GECCOO peuvent contenir des aspects comportementaux (non levée de certaines classes d'exception, séquences d'invocations autorisées, mode d'accès à certaines ressources protégées ...). La prise en compte de ces propriétés nécessite d'étendre l'aspect observationnel lié au raffinement, et les obligations de preuve nécessaires. En effet certaines propriétés comportementales ne s'héritent pas par raffinement, et nécessitent des vérifications à chaque niveau du raffinement. Le but sera donc d'étudier les conditions de raffinement des propriétés de sécurité de la tâche 1. Ces études seront basées pour partie sur les travaux et outils des partenaires, dans le cadre de développements B basés sur la notion d'événements. Les outils développés par l'équipe VaSCo permettent de représenter des comportements, décrits par des événements sous forme de gardes/actions, par des systèmes de transitions [13]. Le but est de pouvoir utiliser la représentation explicite du comportement pour simplifier la preuve, en particulier pour synthétiser certains invariants de raffinement [33, 39]. L'équipe TFC a défini un raffinement de systèmes de transition qui préserve l'ensemble des propriétés exprimables en PLTL (Propositional Linear Temporal Logic) n'utilisant pas l'opérateur "Next" [10, 23]. Ce raffinement intègre la préservation de ces propriétés sous hypothèses d'équité [11]. A partir de cette notion de raffinement, l'équipe TFC a défini des méthodes de vérification de propriétés combinant preuve et model-checking pour une classe des propriétés PLTL. Ces méthodes consistent d'une part à effectuer des vérifications sur chaque partie du modèle obtenue par raffinement [29] et d'autre part à effectuer des vérifications de propriétés raffinées [9]. Ces résultats ont été étendus au cas des systèmes infinis paramétrés.

Nous étudierons l'adaptation de ces résultats aux besoins de vérification des propriétés de sécurité de la classe d'applications visée dans cette ACI.

B2-3 Génération et résolution d'obligations de preuve

Participants LogiCal,CASSIS,Lemme

La tâche 1 permet de produire un code Java annoté par des spécifications JML sous forme de pré-post conditions et d'invariants ; la tâche 2 établit des conditions sous lesquelles un raffinement préserve la spécification. Pour garantir la correction du code par rapport aux objectifs de sécurité, il faut produire des obligations de preuve suffisantes pour que les méthodes satisfassent les conditions et préservent les invariants puis résoudre les conditions de correction produites par cette étape ou celle de raffinement. C'est l'objectif de la tâche 3.

Le but est que les obligations engendrées puissent être traitées principalement de manière automatique mais également de manière interactive. Une certaine latitude est offerte dans le choix de la modélisation des programmes objet, or celle-ci a une influence importante sur la facilité à résoudre les propriétés engendrées. Dans le cadre de ce projet, une collaboration étroite entre les équipes responsables du modèle et de la génération des obligations et celles spécialistes des techniques de résolutions automatiques permettra des avancées significatives.

Cette tâche s'articule en deux activités, l'une plus axée sur le modèle et son impact sur la tractabilité des obligations engendrées, l'autre orientée vers les techniques de résolution.

Génération d'obligations de preuve L'approche développée dans l'outil Krakatoa est modulaire. Le programme Java et les annotations JML sont traduits en une théorie qui modélise la structure des classes et des objets et en un ensemble de programmes exprimés dans le langage d'entrée de l'outil Why (un sous-ensemble de ML comportant des références, des exceptions et des fonctions d'ordre supérieur). Les programmes de Why sont annotés par des pré et post-conditions, assertions, variants et invariants de boucles exprimés dans une logique du premier ordre paramétrable par des sortes et prédicats arbitraires. L'outil Why assure la génération des obligations de preuve en utilisant une première phase de propagation des annotations par un calcul de plus faible précondition puis l'identification des lemmes à prouver par l'utilisateur et finalement la construction d'une validation sous la forme d'un programme fonctionnel exécutable certifié. Les obligations de preuves de Why peuvent être traduites vers différents démonstrateurs (actuellement PVS, Coq et haRVey) et Why accepte également en entrée des programmes d'un sous-ensemble de C. C'est donc un outil très générique.

La forme des obligations de preuve engendrées va dépendre du choix de traduction du programme Java/JML vers le programme Why annoté. Cette traduction nécessite d'expliciter la sémantique de Java et de JML qui peut présenter des points subtils. Elle demande également de choisir un modèle de la mémoire. Dans l'état actuel les variables de la pile sont représentées par des variables logiques qui ne peuvent être aliasées (ce qui est cohérent avec le mode de passage par valeur de Java), par contre les objets sont représentés par des adresses et une variable spéciale représente le tas. Finalement la traduction repose sur des choix méthodologiques, nous avons par exemple modélisé l'accès dans un tableau comme une fonction totale qui ne lève pas d'exception correspondant à un accès en dehors des bornes. Cependant chaque utilisation de cette fonction est protégée par une assertion stipulant que la valeur accédée est bien dans les bornes. Cette approche interdit de traiter un programme dans lequel une erreur d'accès serait intentionnellement réalisée pour être ensuite rattrapée ; un tel style de programmation nous a paru peu conforme aux principes de développement de logiciel critique. Notre choix a pour conséquence d'améliorer la lisibilité des obligations de preuve.

Nous nous proposons d'axer notre travail d'une part sur l'extension des propriétés de programmes garanties, d'autre part sur l'exploration de méthodes permettant de simplifier le modèle d'accès aux objets dans le tas lorsque l'on peut garantir l'absence d'aliasing.

Actuellement, les outils de preuves tels que Jive, LOOP, Krakatoa et Jack ne garantissent que la correction partielle des méthodes mutuellement récursives. On prouve la correction du corps d'une méthode en supposant toutes les méthodes des classes du package correctes. Tout comme on garantit la terminaison des boucles while par la mise en évidence d'un variant, il convient de trouver une annotation naturelle des méthodes permettant de garantir la terminaison. Why traite déjà la terminaison d'une fonction récursive par l'utilisation d'un ordre bien fondé. Cette approche pourrait se généraliser au cas de fonctions mutuellement récursives. Au niveau d'un langage objet, il est préférable de poursuivre une approche où chaque méthode est prouvée de manière indépendante mais où des assertions complémentaires faisant intervenir un ordre bien fondé global seront engendrées.

Un autre point plus technique concerne les problèmes de dépassement de capacité sur les types numériques de base. Deux approches sont connues pour prendre en compte le modèle réel : soit on modélise l'arithmétique modulo qui servira alors de modèle aux opérations Java, c'est l'approche qui a été utilisée dans LOOP. Il faut alors relier la représentation concrète et la traduction vers des entiers mathématiques ; soit on interprète les entiers machines dans le modèle idéal des entiers relatifs non bornés et on ajoute automatiquement en précondition les propriétés qui garantissent l'absence de dépassement. Nous comptons expérimenter ces deux pistes pour comparer la nature et difficulté des obligations de preuve engendrées dans les cas courants.

Un problème essentiel est la difficulté de résolution des obligations de preuve mettant en jeu un modèle mémoire avec des fonctions d'accès et de mise à jour. Ce problème peut s'attaquer de différentes manières. On peut élaborer des procédures de décision, en particulier en liaison avec l'outil haRVey (cf la partie résolution d'obligations de cette tâche). On peut également et de manière complémentaire explorer des modélisations alternatives de la mémoire. Actuellement, Krakatoa, Jive ou LOOP utilisent pour représenter les objets, un tas unique global, qui associe à chaque adresse un ensemble de valeurs correspondant aux différents champs de la mémoire. Une représentation alternative, explorée par exemple dans [14, 37] utilise un modèle alternatif qui introduit autant de "tas" locaux que de champs d'objets. Ainsi la mémoire est structurée en tableaux associés à des champs de classe et indexés par les adresses mémoire. Un accès ou une mise à jour à la valeur $A.f$ correspond à l'accès ou la mise à jour du tableau $f[A]$. Ce modèle est supposé être plus local. En effet lorsqu'on raisonne avec un tas global, si on met à jour le champs f d'un objet alors la variable de tas est mise à jour, donc lorsqu'on accède à un autre champs g de cet objet ou d'un autre objet, il faudra prouver qu'il n'est pas modifié. Par contre si on utilise une variable par champs d'objet, seule la variable correspondant au champs f est modifiée, donc la préservation de la valeur du champs g est obtenue sans preuve. Le défaut de cette approche est qu'elle complexifie le modèle des programmes objets, et qu'elle pourrait compromettre sa modularité. Cependant, les clauses JML permettent de spécifier le sous ensemble de champs qui sont possiblement modifiés. D'autre part, une analyse statique du code des méthodes a déjà lieu pour préciser les exceptions levées et si le tas est possiblement modifié ; ces techniques doivent pouvoir être affinées pour prendre en compte un modèle dans lequel on découpe le tas suivant les champs.

Il est possible de relier le *modèle mémoire* où chaque objet est représenté comme une adresse dans le tas à un *modèle fonctionnel* des objets dans lequel chaque valeur est représentée comme une structure de graphe possédant éventuellement des boucles. Il est beaucoup plus simple de raisonner directement sur ces valeurs fonctionnelles que sur la notion de pointeurs, on peut prendre comme exemple la notion de liste. Cependant, représenter une variable d'objet correspondant à une adresse par une variable ayant pour valeur l'objet fonctionnel accessible à partir de cette adresse n'est pas toujours possible. En effet, si deux variables différentes permettent d'accéder à la même adresse, une modification de l'une des variables peut affecter la valeur de la seconde, mais cette information est perdue dans l'interprétation fonctionnelle. Cependant, des analyses statiques ou des préconditions spécifiques exprimables en JML peuvent assurer l'absence de tels alias et ainsi permettre de raisonner sur le objets avec un plus haut niveau d'abstraction.

Résolution d'obligations de preuve Notre objectif est de développer des procédures de décision pour résoudre le problème de la satisfiabilité dans les théories pertinentes pour le développement de programmes (tableaux, listes, arithmétique et leur combinaison). Ces procédures s'appuient sur les règles d'inférence et de contrôle des démonstrateurs du premier-ordre. Plus précisément notre objectif est d'étendre les théories existantes (par exemple la théorie des tableaux) en identifiant des extensions (comme l'introduction de pointeurs sans alias) et en mettant en place des procédures de décision adaptées. La méthodologie générale pour établir ces résultats est de montrer qu'une application exhaustive des règles d'inférence d'un calcul du premier-ordre correct et complet pour la réfutation (par exemple un calcul de superposition) termine pour la théorie considérée. Les détails de cette approche peuvent se trouver dans [2].

Notre approche repose implicitement sur l'existence d'un catalogue de théories qui sont pertinentes pour l'analyse de programmes. Tout le monde s'accorde à reconnaître que certaines propriétés de programmes sont plus simples à vérifier que d'autres. Il est important de donner des bases solides à cette affirmation. Pour cela, on mettra en évidence une hiérarchie de théories qui peuvent s'utiliser pour exprimer des analyses de programmes de complexité croissante. Cette hiérarchie devrait nous servir de "feuille de route" pour la synthèse, l'extension et la combinaison de procédures de décision.

Le problème de satisfiabilité est défini pour une formule qui est une conjonction de littéraux clos. Un point important est d'associer les procédures de décision à des modules capables de manipuler de grosses formules avec une structure propositionnelle quelconque et qui peuvent éventuellement contenir des quantificateurs. En effet, les formules à montrer pour la vérification de programmes sont souvent très grosses ; les convertir en forme normale conjonctive et utiliser un démonstrateur du premier ordre est souvent vain. Il y a donc deux objectifs à atteindre. Le premier est d'intégrer étroitement un solveur de satisfiabilité propositionnel puissant (SAT) et la procédure de décision pour une théorie donnée. Le solveur SAT prendra en charge efficacement

la structure propositionnelle de la formule close à traiter et la procédure de décision éliminera les cas qui sont cohérents de manière propositionnelle mais non valides dans le domaine de la théorie sous-jacente. Le second objectif est d'intégrer dans la résolution de SAT et la procédure de décision du premier ordre un mécanisme efficace pour traiter les quantificateurs de la formule à valider. Le système haRVey [24] a été conçu dans ce but. Il repose sur l'intégration d'une bibliothèque de BDD qui prend en charge la structure propositionnelle de la formule, un démonstrateur par superposition qui permet d'implanter des procédures de décision pour le problème de satisfiabilité pour différentes théories pertinentes et un pré-traitement qui abstrait une formule contenant des quantificateurs en une formule close en enrichissant la théorie sous-jacente pour prendre en compte ces quantificateurs.

B2-4 Détection d'erreurs

Participants TFC,Cassis,Lemme

Validation du modèle par évaluation des comportements et génération de contre-exemples L'objectif de cette tâche est de définir et développer des techniques de vérification du modèle fondées sur l'évaluation des comportements spécifiés et la génération de contre-exemples.

Il s'agit de pouvoir compléter les vérifications de cohérence réalisées par les outils de preuve en s'appuyant sur des technologies issues de la génération automatique de tests à partir du modèle (outils du LIFC BZ-Testing-Tools). Ces technologies s'appuient sur une transformation du modèle formel en un système de contraintes équivalent, qui est ensuite exploité pour exhiber l'ensemble des comportements du système modélisé. Dans le cas du test, cela permet de générer des séquences de tests fonctionnels aux limites, garantissant une couverture de tous les comportements et de tous les états limites. On désigne par comportement, l'activation d'une certaine post-condition, et par états limites un état du système ou des variables prennent des valeurs extrêmes (minimum, maximum), de leur sous-domaines de définition. Ces techniques seront revisitées pour vérifier le modèle et exploiter les informations issues de la preuve, en particulier les obligations de preuve non prouvées. Ainsi, ce module traitera la négation de ces obligations de preuve non-prouvées comme des objectifs de test, permettant d'exhiber, si cela est possible, des contre-exemples montrant l'incohérence des spécifications. L'évaluation des comportements permettra de détecter certaines mauvaises caractéristiques de la formalisation : invariant trop faible ou trop fort, préconditions trop fortes, postconditions non atteignables, en particulier. Ces approches sont totalement complémentaires des techniques de preuve connues aujourd'hui. Pour les porteurs de ce projet, c'est cette complémentarité qui contribuera à repousser les limites actuelles de la vérification des modèles formels.

L'outil développé dans cette tâche appuiera sur les technologies à contraintes du LIFC et possédera les caractéristiques suivantes :

- Il prendra en entrée les spécifications et annotations formalisées ainsi que l'ensemble des obligations de preuve non-résolue ;
- Il engendrera des contre-exemples relatifs à des obligations de preuve non-prouvées ;
- Il analysera des comportements non-atteignables en identifiant et surlignant des parties de l'invariant susceptibles d'être trop faibles ou trop fortes, de même pour les pré- et post-conditions.

Les programmes sont rarement corrects la première fois qu'ils sont écrits. Habituellement il est nécessaire d'affiner le code originel, dans le cas par exemple où le programmeur a omis de considérer certaines situations. Dans cette phase le programmeur est plus intéressé par la découverte d'erreurs dans son programme que par la preuve de correction. De plus, établir des invariants de boucle est une tâche complexe qui prend du temps et qu'il est préférable de ne commencer que lorsqu'on a confiance dans la correction de son programme. Dans cet objectif, la simulation symbolique semble une solution prometteuse. Seules les pré et post-conditions sont nécessaires pour que cette technique fonctionne. En particulier, la simulation symbolique produira des obligations de preuve (une pour chaque chemin d'exécution de longueur finie) sans nécessiter que le programmeur explicite les invariants de boucle, la boucle étant dépliée un nombre fini de fois. Cette approche permet de mettre en évidence des erreurs. Comme l'évaluation symbolique produit des obligations de preuve, celles-ci doivent être traitées par des techniques de démonstration automatique. La différence avec les techniques de preuve utilisée pour établir la correction des programmes est que nous cherchons plutôt à réfuter les obligations de preuve et à produire des contre-exemples qui expliquent les causes de l'échec. L'activité est difficile si on cherche à donner la justification à un niveau assez abstrait pour être compréhensible par un humain. Notre objectif est d'étendre haRVey dans cette direction.

Génération de code certifié Après avoir spécifié et éventuellement raffiné notre politique de sécurité, nous voulons exécuter l'application de manière sûre. Dans la tâche 1, nous avons établi une traduction de la politique de sécurité en annotations JML. En utilisant les outils JML existants, il est possible de produire une implantation défensive qui vérifie dynamiquement toutes les propriétés spécifiées dans les assertions.

Cependant l'usage de tests dynamiques ralentit le programme, il est donc souhaitable d'éviter d'effectuer des tests superflus. Nous utiliserons les outils de preuve afin de vérifier statiquement quelles assertions sont satisfaites, et de ne pas introduire de vérification dynamiques pour ces dernières.

B2-5 Chaîne de développement

Participants Cassis,TFC,Lemme,LogiCal,VasCo

Cette tâche est une tâche transversale qui a pour objectif d'étudier l'intégration des différentes techniques développées au sein du projet GECCOO pour former un environnement de développement de programmes à fortes contraintes de sécurité. Il est probable que certains résultats obtenus dans chacune des tâches devront être adaptés pour s'insérer harmonieusement dans une chaîne de développement.

Suivant les moyens humains qui seront attribués à cette action, il sera possible de construire des prototypes proposant tout ou partie de cette chaîne. Les candidats à servir de base à ces prototypes sont les systèmes Krakatoa et Jack qui traitent des programmes Java annotés en JML en interaction avec des démonstrateurs automatiques ou interactifs. Si les objectifs de ces deux outils se rejoignent, les choix de design sont différents. Jack, développé dans un contexte industriel dispose d'une interface permettant de relier le code source et les obligations de preuve qui sont présentées à l'utilisateur sous une forme proche de la spécification initiale. Il produit actuellement des obligations de preuve qui sont traitées par le démonstrateur de l'atelier B, ce dernier dispose de plusieurs procédures de preuve automatique et peut être en principe dirigé de manière interactive même si cette dernière activité requiert une expertise certaine. Krakatoa est lui construit sur une chaîne de logiciels libres (Krakatoa-Why-Coq) que les membres du projet peuvent adapter à leurs besoins, cet outil repose également sur une politique de production de "certificats" pour les étapes de preuve qui sont réalisées par la machine, ce qui permet de se prémunir d'erreurs de programmation ou de conception dans les outils eux-mêmes.

La collaboration entre les deux outils au sein de ce projet permettra de confronter les approches en terme de modèles ou de génération d'obligations de preuve. De plus les résultats ou les études de cas peuvent être facilement partagés. L'effort autour de Jack veille à préserver une interface conviviale et facilement utilisable. Cet aspect n'est pas développé dans Krakatoa pour lequel l'utilisateur interagit au niveau de l'assistant Coq.

Jack ou Krakatoa forment le cœur de la chaîne de développement autour de laquelle vont pouvoir venir se greffer des outils de génération de spécification JML à partir de politiques de sécurité (tâche 1), des outils de démonstration ou d'analyse automatique spécialisé basés sur haRVey (tâche 3) ou encore les outils de génération de tests par contraintes (tâche 4). La mise en œuvre en pratique de la tâche 2 de raffinement dans un environnement demande un effort de développement qui semble au delà des possibilités de cette action.

B3 – Résultats attendus :

On détaillera l'échéancier des résultats et réalisations intermédiaires et finaux attendus. On précisera les risques scientifiques qui seront pris. On discutera de l'impact potentiel de ce projet sur les scènes européenne et internationale.

Le travail dans le projet s'organisera autour de réunions de l'ensembles des participants, au moins deux fois par an et plus rapprochées au démarrage du projet. Ces réunions auront pour objectif de partager les connaissances des techniques mises en œuvre par chacun des partenaires du projet, puis de suivre et d'orienter l'avancement des travaux et de favoriser les interactions, en particulier le partage des études de cas. L'avancement de chaque tâche donnera lieu à des collaborations bi ou tri-partites plus spécialisées. Le travail donnera lieu à des publications dans les conférences du domaine et à la mise en œuvre de certaines solutions proposées dans des prototypes qui seront expérimentés sur des cas d'étude.

B3-1 Spécifications de haut niveau

Le travail mené sur un langage de spécifications de haut niveau devrait conduire aux résultats suivants :
- Définition d'un langage formel pour exprimer les propriétés de sécurité de haut niveau, avec une sémantique claire et intégré dans le langage JML ;
- Clarification du rôle des invariants dans la spécification objet, formalisation du principe d'encapsulation dans les approches objets, et identification des difficultés pour la vérification formelle ;
- Extension du calcul du raffinement sur lequel repose la méthode B pour prendre en compte la notion d'instance d'objets et certaines particularités du langage Java. Concrètement nous chercherons à réaliser cela en proposant une intégration du langage de spécification (ici JML) et de certaines constructions du langage de programmation (ici Java) pour simuler une telle approche.

Un important choix dans la conception de JML a été sa facilité d'accès. Des expériences ont montré qu'un simple programmeur Java peut comprendre et même écrire des spécifications JML assez rapidement. C'est pour cette raison, et également pour la large palette d'outils disponibles pour JML que les industriels se sont intéressés à l'utilisation de JML comme langage de spécification. Aussi, il est important d'appliquer nos résultats à des cas d'étude réalistes, afin de pouvoir juger sérieusement de la facilité d'écriture de telles spécifications de haut-niveau.

À travers le monde, il y a plusieurs équipes qui travaillent autour de JML (développement du langage, sémantique, outils). Les participants de ce projet font également partie de cette communauté. Un point important est que cette communauté est très ouverte aux contributions extérieures se rapportant à JML, celles-ci sont discutées et peuvent être finalement intégrées au langage standard. On peut donc espérer que nos propositions concernant une extension du langage auront un impact, également sur les groupes qui travaillent sur ou avec JML.

B3-2 Raffinement et programmation orientée objet

Les résultats attendus de la tâche 2 sont :
- Définition de conditions de préservation par raffinement des propriétés de sécurités définies en tâche 1.
- Définition d'une notion de raffinement dans le cadre objet, et clarification de ses liens avec la structuration objet.
- Application de la méthodologie du raffinement à l'implémentation prouvée d'architectures objet génériques (patrons de conception).

Plus fondamentalement, ces travaux ont pour but d'établir des méthodes formelles de conception objet. On espère rendre ainsi les développements formels plus réutilisables, et en particulier, favoriser la réutilisation de composants formellement prouvés. Dans ce cadre, le raffinement est une notion fondamentale. Il permet en effet de masquer l'implémentation des composants, et de factoriser de nombreuses preuves au niveau de la définition des composants (évitant ainsi aux utilisateurs des composants d'avoir à les faire). Mais il faut aussi fournir une notion de généricité que la méthode B n'a pas aujourd'hui, et que la programmation orientée objet procure sous différentes formes. Enfin, à travers ce projet, on espère aussi propager les méthodes formelles dans le domaine de la conception orientée objet qui influence lui-même fortement le génie logiciel aujourd'hui.

B3-3 Génération et résolution d'obligations de preuve

Les résultats attendus sont :
- Technique de preuve de terminaison de méthodes mutuellement récursives compatible avec la modularité ;
- Modèles alternatifs de la représentation mémoire et évaluation de leur impact sur lisibilité et la simplicité de résolution des obligations de preuve ;
- Élaboration de procédures de décision pour des théories spécialisées à l'analyse de programme.

Les trois axes seront étudiés en parallèle. Le travail débutera par une analyse fine des problèmes et des solutions existantes. Les techniques retenues seront mises en œuvre dans des outils et la pertinence des choix sera validée sur des études de cas concrètes.

Les résultats obtenus ont un intérêt pour l'ensemble des outils de preuve de programmes Java spécifiés en JML, qu'ils reposent sur des démonstrations interactives (auquel cas la lisibilité est importante) ou automatiques.

B3-4 Détection d'erreurs

Les résultats attendus sont :
- Élaboration conceptuelle et théorique de la génération de contre-exemples et de l'évaluation de comportements pour valider et vérifier le modèle ;
- Conception de l'outil d'évaluation des comportements et de génération de contre-exemples ;
- Développement d'un prototype de l'outil et validation
- Expérimentation sur une étude de cas et évaluation

B3-5 Chaîne de développement

Les résultats attendus en complément des développements logiciels proposés dans les tâches 1 à 4 sont :
- Consolidation des outils existants, en particulier extensions de leur capacité à communiquer ;
- Amélioration de l'interaction avec l'utilisateur ;

– Intégration d'outils de recherche de contre-exemple ;
– Intégration d'autres techniques et outils développés dans le projet en fonction de leur état d'avancement.

Le projet met en relation des équipes dont l'expertise est diversifiée et complémentaire (preuves automatiques, générateur d'obligations de preuves, modèles objets, raffinement, sécurité des applications JavaCard, génération de tests). Les défis de chacune des sous-tâches ont été identifiés. Vis-à-vis du processus global de développement formel d'applications objet, il est probable que nous n'arriverons pas à proposer une approche complète intégrant à la fois la puissance d'expression des spécifications, la facilité d'utilisation, la réutilisation et l'automatisation des preuves. Néanmoins des avancées significatives sont attendues vis-à-vis de la certification d'applications, qui impose la traçabilité formelle des propriétés exigées, et sur le développement par raffinement jusqu'à du code prouvé. L'intégration progressive même partielle des solutions proposées dans des outils diffusables reposant sur des bases théoriques solides devrait permettre d'aboutir à des environnements très compétitifs sur le plan international.

B4 – Summary (in English) : (1 to 2 pages)

Le Conseil Scientifique pourra solliciter des experts non francophones auxquels sera envoyé l'ensemble des documents. Le présent résumé, entièrement rédigé en anglais, visera à fournir une présentation synthétique de l'ensemble du projet.

This project aims at developing methods and tools for the design of object-oriented systems that require a high degree of security. The methods and tools will be developed to be integrated, *i.e.* together they will form a coherent design method from specification to certified code generation using refinement, simulation, testing and verification techniques. In particular, the project focuses on the design of smart card applications, written in a subset of Java (like JavaCard), annotated with JML specifications. JML, the Java Modeling Language, is a source code level behavioural specification language for Java that is designed to be easy to understand for people with Java knowledge. Several tools exist which manipulate JML annotated programs, *e.g.* to generate documentation or tests (either dynamic or unitary) or to do automatic or interactive verification. The readability of JML specifications and the wide range of tools available for JML make it attractive for industry to integrate formal methods (in the form of JML specifications) in their software development process.

However, experiences with using JML for industrial applications also have revealed several of its shortcomings which this project proposes to attack. First of all, it is important that the requirements on the application can be expressed formally, and at an abstract level, from the beginning of the development process. Subsequently, these specifications can then be refined, until an executable and efficient program has been constructed. These refinement steps can give rise to proof obligations, for which we need to have proof procedures which can handle most of these proof obligations automatically. And since it rarely is the case that a specification and a program are correct from the start, it is important to have testing and simulation techniques to detect errors as soon as possible. Finally, sometimes it is (practically) impossible to prove the correctness of an implementation. Therefore, we want to be able to generate defensive code, which can be validated using the testing and simulation techniques.

In all this work, the notion of proof is central. Each step in the development process should be formally justified and whenever possible, the justification should be mechanised with a computer. Because of undecidability results, in general it is impossible to prove automatically the correctness of all required properties. To show them by hand is theoretically feasible, however in practice this often is too much work. Therefore, it is necessary to (1) develop automatic methods tailored to the specific class of problems encountered when adressing security properties of object-oriented programs and (2) to combine interactive and automatic techniques. In particular, one can combine static or dynamic tests and verification techniques in order to eventually obtain correct executable code.

The project will focus on four main topics, which correspond to the different development steps for software with high degree security requirements :

- high-level specifications ;
- refinement and object oriented programming ;
- generation and resolution of proof obligations ; and
- error detection.

A fifth task focuses on integration of the different components in an environment for developing certified programs.

For each of the topics, we will briefly discuss what we expect to achieve within this project.

High-level specifications We shall identify a high-level specification language for describing common security policies, for example :

- access control : a command is not executed except in authenticated mode ;
- integrity : data cannot be modified except in authenticated mode by a trusted partner ;
- confidentiality : data can only be accessed in authenticated mode by a trusted partner ;
- availability of service : a request is always followed by an answer ;
- safety : exceptions of a certain kind are never raised ; and
- methods are only called in a particular order, methods are only called a limited number of times etc.

Many of these properties cannot directly be expressed on the level of methods, so therefore we need a higher level of specification, which can be used to restrict the interaction between different methods (and objects). We define a semantics for this language, and clarify the role of invariants and encapsulation in object-oriented modeling in order to compose specifications in a modular way without redoing the proofs.

Refinement and object oriented programming The specification language that we will define should be suitable for development by refinement. The refinement proces should also be integrated with the object-

oriented development model, which based on inheritance and overriding. A first objective is to study refinement at the level of classes and methods. Next, we will extend the work to refinement of components and the construction of specifications by composition of specifications. An important issue is to study under which conditions security properties are preserved by refinement.

Generation and resolution of proof obligations The correctness of refinement steps and possibly of the code is established by generating and subsequently solving proof obligations. The complexity of properties depends on the theory in which they are expressed and on our requirements on what has to be proved. For example, existing tools only ensure partial correctness of recursive methods, and often do not check possible overflows in arithmetic computations. Within this project, we will aim at handling this kind of properties. However, in many cases one might only be interested in exception propagation or the non-modification of some fields. For such more restricted properties, we will developed appropriate automated techniques. In general, it is necessary to relate interesting security properties and proof techniques, in order to improve the automatic resolution of proof obligations.

Error detection It is also possible to use specification annotations in the code in order to do early error detection. If at some control point in the program, a proof obligation cannot be solved automatically, it is possible to replace verification by testing. There are different ways in which this can be done : one can generate adequate inputs to reach the control point or one can introduce dynamic tests in the generated code. Partial evaluation and simulation using constraint resolution may also be interesting methods to detect errors in the code or in the specification at an early stage.

Environment The teams collaborating in this project have complementary skills in the domains of security, modeling object oriented programs, and interactive and automatic program verification. Collaborations already exist : the tool Why for program verification [26, 25] has been adapted to produce proof-obligations that can be solved by the automatic solver haRVey [24]. The people at LIFC and LORIA are part of a common INRIA team CASSIS. They collaborate with the VasCo team in the national RNTL project B-OM. This project studies optimisation of generated code using the B-method for execution on a smartcard. Lemme and LogiCal participate in the european IST project VERIFICARD[8] which is in particular concerned with verification of JavaCard applets using JML specifications.

[8] `http://www.verificard.org`

Références

[1] Jean-Raymond Abrial. *The B-Book, assigning programs to meaning*. Cambridge University Press, 1996.

[2] Alessandro Armando, Sylvio Ranise, and Michaël Rusinowitch. A rewriting approach to satisfiability procedures. *Information and Computation*, 2002. À paraître.

[3] Site web de l'atelier B. `http://www.atelierb.societe.com`.

[4] Site web de B Tool. `http://www.b-core.com/btool.html`.

[5] Ralph-Johan Back, Anna Mikhajlova, and Joakim von Wright. Class refinement as semantics of correct object substitutability. Technical Report 333, Turku Center for Computer Science, 2000.

[6] Ralph-Johan Back and Joakim von Wright. *Refinement Calculus : A Systematic Introduction*. Springer-Verlag, 1998.

[7] Thomas Ball and Sriram K. Rajamani. Slam, 2002. `http://research.microsoft.com/slam`.

[8] D. Bartetzko, C. Fischer, M. Möller, and H. Wehrheim. Jass – Java with Assertions. In K. Havelund and G. Roşu, editors, *Electronic Notes in Computer Science*, volume 55(2). Elsevier Science BV, 2001.

[9] F. Bellegarde, C. Darlotand J. Julliand, and O. Kouchnarenko. Reformulation : a Way to Combine Dynamic Properties and B Refinement. In *FME 2001 ("Formal Methods Europe")*, volume 2021 of *LNCS*. Springer-Verlag, 2001.

[10] F. Bellegarde, J. Julliand, and O. Kouchnarenko. Ready-simulation is not Ready to Express a Modular Refinement Relation. In *Fundamental Aspects of Software Engineering 2000, FASE'2000*, volume 1783 of *LNCS*. Springer-Verlag, 2000.

[11] F. Bellegarde, J. Julliand, and O. Kouchnarenko. Synchronized Parallel Composition of Event Systems in B. In *2nd International Conference of B and Z Users, ZB2002*, volume 2272 of *LNCS*. Springer-Verlag, 2002.

[12] Joachim van den Berg and Bart Jacobs. The LOOP compiler for Java and JML. In T. Margaria and W. Yi, editors, *Tools and Algorithms for the Construction and Analysis of Software*, volume 2031 of *LNCS*, pages 299–312. Springer-Verlag, 2001.

[13] Didier Bert and Francis Cave. Construction of Finite Labelled Transition Systems from B Abstract Systems. In *Integrated Formal Methods, IFM2000*, volume 1945 of *LNCS*, pages 235–254. Springer-Verlag, novembre 2000.

[14] Richard Bornat. Proving pointer programs in Hoare logic. In R. Backhouse and J. N. Oliveira, editors, *Mathematics of Program Construction, 5th International Conference, MPC'2000*, volume 1837 of *LNCS*. Springer-Verlag, 2000.

[15] C. Breunesse, B. Jacobs, and J. van den Berg. Specifying and Verifying a Decimal Representation in Java for Smart Cards. In H. Kirchner and C. Ringeissen, editors, *Proc. Int. Conf. on Algebraic Methodology And Software Technology AMAST'2002*, number 2422 in LNCS, pages 304–318. Springer-Verlag, 2002.

[16] Martin Büchi and Wolfgang Weck. The greybox approach : When blackbox specifications hide too much. Technical Report 297, Turku Center for Computer Science, August 1999.

[17] Lilian Burdy, Yoonsik Cheon, David Cok, Michael Ernst, Joe Kiniry, Gary T. Leavens, K. Rustan M. Leino, and Erik Poll. An overview of JML tools and applications. Technical Report NIII-R0309, Dept. of Computer Science, University of Nijmegen, 2003.

[18] Lilian Burdy, Jean-Louis Lanet, and Antoine Requet. Jack : Java Applet Correctness Kit. `http://www.gemplus.com/smart/r_d/trends/jack.html`.

[19] Lilian Burdy and Antoine Requet. Jack : Java Applet Correctness Kit. In *Gemplus Developers Conference GDC'2002*, 2002.

[20] Lilian Burdy, Antoine Requet, and Jean-Louis Lanet. Java applet correctness : a developer-oriented approach. Soumis, 2003.

[21] N. Cataño and M. Huisman. Formal specification and static checking of Gemplus's electronic purse using ESC/Java. In L.-H. Eriksson and P.A. Lindsay, editors, *Proc. Formal Methods Europe FME'02*, number 2391 in LNCS, pages 272–289. Springer-Verlag, 2002.

[22] N. Cataño and M. Huisman. Chase : a Static Checker for JML's Assignable Clause. In L.D. Zuck, P.C. Attie, A. Cortesi, and S. Mukhopadhyay, editors, *Verification, Model Checking and Abstract Interpretation (VMCAI '03)*, number 2575 in LNCS, pages 26–40. Springer-Verlag, 2003.

[23] C. Darlot, J. Julliand, and O. Kouchnarenko. Refinement Preserves PLTL Properties. In *3nd International Conference of B and Z Users, ZB2003, à paraître*. Springer-Verlag, 2003.

[24] David Déharbe and Sylvio Ranise. haRVey, 2002. http://www.loria.fr/~ranise/haRVey.

[25] J.-C. Filliâtre. Why : a multi-language multi-prover verification tool. Submitted to FME 2003, 2003.

[26] Jean-Christophe Filliâtre. The Why certification tool. http://why.lri.fr/.

[27] Bart Jacobs and Erik Poll. Coalgebras and monads in the semantics of Java. *Theoretical Computer Science*, 291 :329–349, 2002.

[28] Site web de JML. http://www.jmlspecs.org.

[29] J. Julliand, P-A. Masson, and H. Mountassir. Vérification par model-checking modulaire des propriétés dynamiques introduites en B. *Technique et Science Informatiques*, 20(7), 2001.

[30] The Krakatoa team. The Krakatoa proof tool, 2002. http://www.lri.fr/~marche/krakatoa.

[31] M. Labeth, J. Meyer, P. Müller, and A. Poetzsch-Heffter. Formal Verification of a Doubly Linked List Implementation : A Case Study Using the JIVE System. Technical Report 270, Fernuniversität Hagen, 2000.

[32] Gary T. Leavens, Albert L. Baker, and Clyde Ruby. Preliminary design of JML : A behavioral interface specification language for Java. Technical Report 98-06i, Iowa State University, 2000.

[33] J. Lebray. Modélisation de systèmes en B : Proposition de guides méthodologiques pour la décomposition d'événements. Rapport de DEA, Institut National Polytechnique de Grenoble, France, 2000.

[34] K.R.M. Leino, G. Nelson, and J.B. Saxe. ESC/Java user's manual. Technical Report SRC 2000-002, Compaq System Research Center, 2000.

[35] The LOOP Team. Loop project. http://www.cs.kun.nl/~bart/LOOP.

[36] Claude Marché, Christine Paulin, and Xavier Urbain. The Krakatoa tool for JML/Java program certification. Available at http://krakatoa.lri.fr, 2003.

[37] Farhad Mehta and Tobias Nipkow. Proving pointer programs in higher-order logic. In *Automated Deduction - CADE-19*, LNCS. Springer-Verlag, 2003.

[38] J. Meyer, P. Müller, and A. Poetzsch-Heffter. The JIVE system—implementation description. Available from http://www.informatik.fernuni-hagen.de/pi5/publications.html, 2000.

[39] J. Nahoum. Outils d'assistance à la construction de systèmes dans la méthode B. Rapport de DEA, Institut National Polytechnique de Grenoble, France, 2001.

[40] Erik Poll, Joachim van den Berg, and Bart Jacobs. Formal specification of the JavaCard API in JML : the APDU class. *Computer Networks*, 36(4) :407–421, 2001.

[41] Marie-Laure Potet. Spécifications et développements structurés dans la méthode B. In D. Bert, V. Donzeau-Gouge, and H. Habrias, editors, *Développement rigoureux de logiciel avec la méthode B*, volume 22. Technique et Science Informatiques, 2003.

[42] Marie-Laure Potet and Yves Rouzaud. Composition and Refinement in the B-Method. In D. Bert, editor, *Proceedings of the Second International B Conference*, volume 1393 of *LNCS*. Springer-Verlag, 1998.

[43] Mark Utting. *An Object-Oriented Refinement Calculus with Modular Reasoning*. PhD thesis, University of New South Wales, Kensington, Australia, 1992.

Action Concertée Incitative

SÉCURITÉ INFORMATIQUE

Descriptif complet du projet

C – MOYENS FINANCIERS ET HUMAINS DEMANDÉS PAR CHAQUE ÉQUIPE

Comme indiqué dans les tableaux ci-dessous, on distinguera

- *les financements via le Fonds National pour la Science qui peuvent inclure*
  - *du fonctionnement*
  - *de l'équipement*
  - *des mois de personnel temporaire (CDD) pour un montant ne pouvant excéder 50% du financement total attribué. La durée du ou des contrat(s) prévus, qui ne peuvent excéder 24 mois chacun, sera précisée.*
- *les moyens demandés aux organismes de recherche qui peuvent inclure*
  - *des postes de post-doc*
  - *des demandes de délégation ou détachement pour des enseignants-chercheurs*
  - *des accueils de chercheurs étrangers*
- *les demandes d'allocations de recherche*

Les diverses possibilités concernant l'attribution de moyens pour recruter ou accueillir des personnels seront globalement très limitées pour l'ensemble des ACI. Leurs demandes devront donc être particulièrement justifiées. Si les bénéficiaires de ces demandes sont connus ou pressentis, les CV correspondants seront joints à la présente demande.

Dans le cas des moyens alloués par les organismes, il n'est pas nécessaire de préciser à quel organisme (CNRS ou INRIA) ces moyens seront demandés, sauf cas particulier à expliciter. Ces moyens seront en effet répartis globalement au niveau de l'ACI, en tenant compte bien sûr des règles et contraintes propres à chaque organisme.

On présentera une justification scientifique des moyens demandés pour chacune des équipes impliquées dans le projet.

Action Concertée Incitative

SÉCURITÉ INFORMATIQUE

Descriptif complet du projet

C1 - Demandes effectuées dans le cadre de l'ACI pour le présent projet :

Nom de l'équipe ou du laboratoire : LogiCal

Moyens demandés dans le cadre de la présente ACI (en KETTC) :

Financements via le Fonds National de la Science :

	2003	2004	2005	Total
Équipement	5	5	5	15
Fonctionnement (dont CDD décrits ci-dessous)	20	20	20	60
Total / année	25	25	25	75

Dépenses de personnels (CDD) :

Nature de l'emploi (post-doc, ingénieur, assistant-ingénieur,...)	
Durée de l'emploi (en mois)	
Coût total de l'emploi	

Financements via les organismes de recherche :

	2003–2004	2004–2005	2005–2006	Total
Nombre de post-docs (préciser pour chaque demande la durée en mois)				
Nombre d'accueils de chercheurs étrangers (préciser pour chaque demande la durée en mois)				
Nombre d'accueils en délégations ou détachementsk		1	1	2

Allocations de recherche :

	2003–2004	2004–2005	2005–2006	Total
Nombre d'allocations de recherche débutant en :		1		1

Justifications scientifiques de l'ensemble des demandes :

L'expérimentation des méthodes de preuve requiert des moyens de calculs importants en mémoire et rapidité. Étant donnée l'évolution rapide des matériels, il est important de pouvoir le renouveler régulièrement. Le fonctionnement est destiné à financer la participation aux conférences internationales du domaine, à fi-

nancer des visites de courtes durées dans les sites internationaux travaillant sur des sujets en relation avec le projet GECCOO et à financer des indemnités de stage (maîtrise et DEA).

Nom de l'équipe ou du laboratoire : CASSIS

Moyens demandés dans le cadre de la présente ACI (en KETTC) :

Financements via le Fonds National de la Science :

	2003	2004	2005	Total
Équipement	5	5	5	15
Fonctionnement (dont CDD décrits ci-dessous)	20	20	20	60
Total / année	25	25	25	75

Dépenses de personnels (CDD) :

Nature de l'emploi (post-doc, ingénieur, assistant-ingénieur,...)	
Durée de l'emploi (en mois)	
Coût total de l'emploi	

Financements via les organismes de recherche :

	2003–2004	2004–2005	2005–2006	Total
Nombre de post-docs (préciser pour chaque demande la durée en mois)	1 12 mois			1 12 mois
Nombre d'accueils de chercheurs étrangers (préciser pour chaque demande la durée en mois)	1 mois	1 mois	1 mois	3 mois
Nombre d'accueils en délégations ou détachements	1			1

Allocations de recherche :

	2003–2004	2004–2005	2005–2006	Total
Nombre d'allocations de recherche débutant en :				

Justifications scientifiques de l'ensemble des demandes : L'accueil en détachement devrait permettre à Laurent Vigneron [9] de participer au projet et d'y apporter ses compétences dans les domaines de la preuve automatique de théorèmes et de la vérification de protocoles de sécurité. Le chercheur étranger est David Déharbe[10], co-auteur de l'outil haRVey, et maître de conférences au Brésil. Ses visites régulières permettront de continuer le développement de cet outil.

[9] Un CV et une liste de publications sont disponibles : `http://www.lri.fr/~paulin/ACI/cv-vigneron.ps` et `http://www.lri.fr/~paulin/ACI/pub-vigneron.ps`

[10] Un CV est disponible : `http://www.lri.fr/~paulin/ACI/deharbe.ps`

Nom de l'équipe ou du laboratoire : Lemme

Moyens demandés dans le cadre de la présente ACI (en KETTC) :

Financements via le Fonds National de la Science :

	2003	2004	2005	Total
Équipement	6	6	6	18
Fonctionnement (dont CDD décrits ci-dessous)	42	42	12	96
Total / année	48	48	18	114

Dépenses de personnels (CDD) :

Nature de l'emploi (post-doc, ingénieur, assistant-ingénieur,...)	ingénieur
Durée de l'emploi (en mois)	12 mois
Coût total de l'emploi	60

Financements via les organismes de recherche :

	2003–2004	2004–2005	2005–2006	Total
Nombre de post-docs (préciser pour chaque demande la durée en mois)				
Nombre d'accueils de chercheurs étrangers (préciser pour chaque demande la durée en mois)				
Nombre d'accueils en délégations ou détachements				

Allocations de recherche :

	2003–2004	2004–2005	2005–2006	Total
Nombre d'allocations de recherche débutant en :	1			1

Justifications scientifiques de l'ensemble des demandes : Les frais de fonctionnement correspondent aux missions, pour établir la collaboration avec les autres partenaires dans le projet et pour assister aux conférences internationales du domaine.

Un stage de DEA doit débuter en Avril sur les thématiques proposées dans ce projet et pourrait se poursuivre en thèse si une allocation peut lui être attribuée. La demande de bourse de thèse, si elle ne peut être accordée en 2003, sera reportée à 2004.

La demande d'ingénieur, correspond à une situation très particulière. Lilian Burdy[11], chef du projet JACK dans l'équipe méthodes formelles à Gemplus est intéressé à travailler comme ingénieur dans le cadre de cette action. Sa participation est d'un grand apport pour le projet. C'est un spécialiste de la sécurité, de techniques de raffinement (sa thèse réalisée chez Matra portait sur la méthode B), des technologies Java/JML et des méthodes de génération d'obligations de preuve. Il pourra donc intervenir comme expert dans différentes tâches de cette action. Par ailleurs, en poursuivant son travail de développement de l'outil Jack, il pourra en intégrant partiellement les techniques développées dans GECCOO faciliter la cohésion du projet et permettre une expérimentation rapide sur des cas d'étude.

[11] CV accessible `http://www.lri.fr/~paulin/ACI/burdy.rtf`

Nom de l'équipe ou du laboratoire : Laboratoire LSR - équipe VaSCo

Moyens demandés dans le cadre de la présente ACI (en KETTC) :

Financements via le Fonds National de la Science :

	2003	2004	2005	Total
Équipement	12	7	7	26
Fonctionnement (dont CDD décrits ci-dessous)	20	20	20	60
Total / année	32	27	27	86

Dépenses de personnels (CDD) :

Nature de l'emploi (post-doc, ingénieur, assistant-ingénieur,...)	
Durée de l'emploi (en mois)	
Côut total de l'emploi	

Financements via les organismes de recherche :

	2003–2004	2004–2005	2005–2006	Total
Nombre de post-docs (préciser pour chaque demande la durée en mois)		1 post-doc 12 mois		
Nombre d'accueils de chercheurs étrangers (préciser pour chaque demande la durée en mois)				
Nombre d'acceuils en délégations ou détachements				

Allocations de recherche :

	2003–2004	2004–2005	2005–2006	Total
Nombre d'allocations de recherche débutant en :	1			1

Justifications scientifiques de l'ensemble des demandes :

Les frais de fonctionnement demandés dans le cadre de ce projet correspondent aux missions propres au projet et à la participation à des conférences du domaine. Ils incluent aussi des stages (environ 6 mois par an) en lien avec la boite à outils développée au LSR.

Un sujet de thèse est prévu dans le cadre de ce projet. Il portera principalement sur les aspects spécification et développement modulaires. Les travaux développés devront intégrer la vérification compositionnelle et le raffinement modulaire, par analyse de la structuration. Un candidat est pressenti sur le sujet. Il a déjà participé à une étude de cas industrielle de développement par raffinement dans le cadre du projet RNTL BOM. Il a aussi fortement contribué aux outils développés par l'équipe, dans le cadre de ses travaux de magistère et de Dea.

Nom de l'équipe ou du laboratoire : LIFC, équipe TFC

Moyens demandés dans le cadre de la présente ACI (en KETTC) :

Financements via le Fonds National de la Science :

	2003	2004	2005	Total
Équipement	6	6	6	18
Fonctionnement (dont CDD décrits ci-dessous)	25	25	25	75
Total / année	31	31	31	93

Dépenses de personnels (CDD) :

Nature de l'emploi (post-doc, ingénieur, assistant-ingénieur,...)	
Durée de l'emploi (en mois)	
Coût total de l'emploi	

Financements via les organismes de recherche :

	2003–2004	2004–2005	2005–2006	Total
Nombre de post-docs (préciser pour chaque demande la durée en mois)			1 12 mois	1 12 mois
Nombre d'accueils de chercheurs étrangers (préciser pour chaque demande la durée en mois)				
Nombre d'acceuils en délégations ou détachements				

Allocations de recherche :

	2003–2004	2004–2005	2005–2006	Total
Nombre d'allocations de recherche débutant en :	1			1

Justifications scientifiques de l'ensemble des demandes :

Le sujet de thèse portera sur la génération de tests à partir d'objectifs de test qui sont des obligations de preuve non vérifiées et l'utilisation de raffinement afin de maîtriser l'explosion combinatoire du nombre de tests. Plusieurs étudiants de DEA cette année sont des candidats potentiels à une allocation sur ce thème.

Action Concertée Incitative

SÉCURITÉ INFORMATIQUE

Descriptif complet du projet

C2 - Autres soutiens financiers apportés au projet :

On mentionnera les autres actions relatives au projet dans lesquelles l'équipe ou le laboratoire est engagé (projets européens, RNRT, RNTL, autres ACI, ...).

Le LIFC est engagé sur ces sujets dans le projet RNTL DANOCOPS.

Action Concertée Incitative

SÉCURITÉ INFORMATIQUE

Descriptif complet du projet

D - RÉCAPITULATIF GLOBAL DES DEMANDES DU PROJET :

Financements via le Fonds National de la Science :

	2003	2004	2005	Total
Équipement	34	29	29	92
Fonctionnement (dont CDD décrits ci-dessous)	127	127	97	351
Total / année	161	156	126	443

Dépenses de personnels (CDD) :

Nature de l'emploi (post-doc, ingénieur, assistant-ingénieur,...)	Ingénieur
Durée de l'emploi (en mois)	12 mois
Coût total de l'emploi	60

Financements via les organismes de recherche :

	2003–2004	2004–2005	2005–2006	Total
Nombre de post-docs (préciser pour chaque demande la durée en mois)	1 12 mois	1 12 mois	1 12 mois	2 36 mois
Nombre d'accueils de chercheurs étrangers (préciser pour chaque demande la durée en mois)	1 1 mois	1 1 mois	1 1 mois	1 3 mois
Nombre d'accueils en délégations ou détachements	1	1	1	3

Allocations de recherche :

	2003–2004	2004–2005	2005–2006	Total
Nombre d'allocations de recherche débutant en :	3	1		4

Action Concertée Incitative

SÉCURITÉ INFORMATIQUE

Descriptif complet du projet

E - ENGAGEMENT DU COORDINATEUR DU PROJET :

La présente page ne sera remplie que dans la version sous forme papier.

Je soussigné, Christine Paulin, coordinatrice du projet GECCOO, m'engage dans l'hypothèse où le présent projet serait retenu à :

- fournir un rapport d'évaluation à mi-parcours permettant au Conseil Scientifique d'apprécier l'avancement des travaux et la coopération des équipes participantes,
- un rapport à la fin de l'exécution du projet,
- maintenir régulièrement une page web résumant l'ensemble des activités du projet.

Signature du coordinateur du projet :

Visa du Directeur du Laboratoire ou de l'Unité de Recherche auquel appartient le coordinateur du projet :