Actes des journées du GDR GPL 2021

Mireille Blay-Fornarino, Catherine Dubois, Alain Giorgetti, Nikolai Kosmatov

Actes des journées du GDR GPL 2021

Groupement de Recherche « Génie de la Programmation et du Logiciel »

En distanciel, 14-18 juin 2021

Comité d’organisation

- Mireille Blay-Fornarino
- Catherine Dubois
- Frédéric Dadeau
- Alain Giorgetti
- Laure Gonnord
- Loïg Jezequel
- Nikolai Kosmatov
- Yves Ledru
- Responsables des groupes de travail

Editeurs

- Mireille Blay-Fornarino
- Catherine Dubois
- Alain Giorgetti
- Nikolai Kosmatov

Préface

C'est avec grand plaisir que nous vous accueillons pour les douzièmes journées nationales du GDR « Génie de la Programmation et du Logiciel » (GPL) qui se déroulent en distanciel. Malgré les conditions particulières de ces journées, nous nous réjouissons de cette occasion de partager avec toute notre communauté les travaux de ces derniers mois. Il s'agit de nos premières journées en tant que co-directrices du GDR GPL et nous remercions très vivement, Pierre-Etienne Moreau pour avoir animé le GDR GPL ces dernières années et nous avoir aider à prendre sa suite. C'est un honneur pour nous de succéder ainsi à Yves Ledru, Laurence Duchien et Pierre-Etienne Moreau qui, successivement au fil de ces 13 dernières années, ont animé et structuré la communauté au sein de la thématique de la « science du logiciel ».

Les journées nationales sont un temps fort de l'activité de notre GDR, l'occasion de s'enrichir des derniers travaux présentés, d'échanger notamment entre groupes de travail, de découvrir de nouvelles problématiques et de permettre aux jeunes chercheurs de s'ouvrir à la richesse de notre communauté dans un contexte convivial. Plusieurs évènements scientifiques sont co-localisés avec ces journées nationales : la 9ème édition de la conférence en IngénieErie Logicielle (CIEL 2021), ainsi que la 20ème édition de l'atelier francophone sur les Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL 2021).

Cette année, la présentation des posters et les démonstrations d'outils se feront en ligne dans un environnement virtuel en deux dimensions pour favoriser les échanges. Nous remercions Frédéric Dadeau pour avoir permis de maintenir ce temps de partage autour des posters et des outils.

Deux conférenciers invités nous ont fait l'honneur d'accepter notre invitation. Tegawandé F. Bissyandé (Interdisciplinary Centre for Security, Reliability and Trust, Luxembourg) présentera une conférence intitulée « Réparation automatique des logiciels : le rêve et la fantaisie » et Patricia Bouyer-Decitre (Laboratoire Méthodes Formelles, Paris) fera une rétrospective allant « De l'analyse automatique de systèmes temporisés au contrôle de systèmes dynamiques ».

Nous avons porté une attention particulière aux jeunes chercheurs. Nous avons associé l'Ecole des Jeunes Chercheurs en Programmation (EJCP) aux journées nationales avec des leçons données toutes les fins d'après-midi, leçons ouvertes à tous. De plus, pour aider nos jeunes chercheurs à se présenter aux concours CNRS et leur donner confiance et envie d'y participer, pour la première année, Sandrine Blazy et Laurence Duchien, membres du Comité National de la Recherche Scientifique (section 6) animent un atelier où de nombreux témoignages et conseils leur seront délivrés. Nous les remercions vivement pour cette opportunité et la richesse des interventions prévues.

Le GDR GPL a à cœur de mettre à l'honneur les jeunes chercheurs. C'est pourquoi nous décernerons un prix de thèse du GDR pour la neuvième année consécutive. Nous aurons le plaisir de remettre le prix de thèse GPL à Benjamin Farinier pour sa thèse intitulée « Procédures de décision pour l'analyse de vulnérabilité », ainsi que deux accessits, respectivement à Valentin Besnard pour sa thèse intitulée « EMI: Une approche pour unifier l'analyse et l'exécution embarquée à l'aide d'un interpréteur de modèles pilotable » et à Lélio Brun pour sa thèse intitulée « Sémantique mécanisée et compilation vérifiée pour un langage synchrone à flots de données avec réinitialisation ». Le jury chargé de sélectionner le lauréat a été présidé par Pascale Le Gall et Pascal Poizat, que nous remercions tout particulièrement, ainsi que l'ensemble des membres du jury.

Avant de clôturer cette préface, nous tenons à remercier tous ceux qui ont contribué à l'organisation de ces journées nationales, à savoir les responsables de groupes de travail et les membres du comité de direction du GDR GPL. Nous remercions tout particulièrement Alain Giorgetti, Laure Gonnord, Nikolai Kosmatov, Loïg Jezequel et Yves Ledru qui ont été d'un soutien constant pour que ces journées aient lieu.

Mireille Blay-Fornarino et Catherine Dubois

Co-directrices du GDR « Génie de la Programmation et du Logiciel »

Table des matières

Exposés invités

Réparation Automatique des Logiciels: le Rêve et la Fantaisie

Tegawandé Bissyande * [1]

[1] Interdisciplinary Centre for Security, Reliability and Trust [Luxembourg] (SnT) – 6, rue Richard Couhenhove-Kalergi L-1359 Luxembourg, Luxembourg

La correction automatique des bogues, c'est-à-dire l'idée d'avoir des programmes qui corrigent d'autres programmes, est un rêve de longue date qui est de plus en plus embrassé par la communauté du génie logiciel. En effet, malgré les efforts considérables que les développeurs informatiques consacrent à la révision du code et à l'exécution de campagnes de tests logiciels, des erreurs de programmation se glissent, avec quelques fois de graves conséquences. La correction automatique de ces erreurs a récemment fait l'objet d'un certain nombre de travaux potentiellement prometteurs.
Ces techniques, qui ne semblaient initialement valables que pour des cas d'utilisation hypothétiques, ont très récemment été reconnues capables de corriger des bogues dans le monde réel. Par exemple, le projet académique Repairnator, a réussi à montrer que les moteurs de réparation automatique peuvent être compétitifs avec les développeurs humains : les mainteneurs de logiciels libres ont accepté et fusionné des correctifs sans se rendre compte qu'ils avaient été générés automatiquement et soumis par un robot. Au-delà de cet exploit anecdotique, un récent rapport d'expérience sur l'intégration du système de réparation SapFix chez Facebook (un géant de l'industrie du logiciel) suggère définitivement que la réparation automatisée de programmes n'est plus une fiction.

Malheureusement, plusieurs défis fondamentaux empêchent la réparation automatique des bogues d'être largement adoptée dans l'industrie du logiciel. Des études, y compris certaines par notre équipe, ont en effet mis en évidence divers biais impliquant chaque étape de la réparation automatique classique : des hypothèses irréalistes sur la localisation des fautes, l'ajustement excessif de la suite de tests dans la génération des correctifs, l'ajustement excessif au benchmark dans l'évaluation, la concentration sur les bogues simples et triviaux, etc. Ces préjugés finissent par rendre les praticiens et les autres chercheurs méfiants vis-à-vis des promesses de l'APR.
Dans cet exposé, nous ferons le tour sur les hypothèses utilisées dans la littérature, les limites et biais des évaluations, tout en mettant en avant les possibilités et le grand champ de recherche qui mérite exploration.

*Intervenant

De l'analyse automatique de systèmes temporisés au contrôle de systèmes dynamiques

Patricia Bouyer-Decitre * [1]

[1] Laboratoire Méthodes Formelles (LMF) – Institut National de Recherche en Informatique et en Automatique, CentraleSupélec, Université Paris-Saclay, Centre National de la Recherche Scientifique : UMR9021, Ecole Normale Supérieure Paris-Saclay – 4, avenue des Sciences, 91190, Gif-sur-Yvette, France

Nous nous intéresserons au modèle des automates temporisés, qui a été proposé dans les années 1990 pour représenter les systèmes soumis à des contraintes temporelles quantitatives. Nous verrons quelles bonnes propriétés sont satisfaites par ce modèle et en explorerons quelques extensions. Nous présenterons enfin le modèle des automates à "entonnoirs" (*funnel automata*), basé sur les automates temporisés et leurs extensions, qui permettent d'appréhender et résoudre des problèmes de contrôle de systèmes dynamiques.

Cet exposé illustrera comment le domaine des méthodes formelles, et en particulier la thématique du model-checking, peut trouver des applications dans des domaines voisins tels que la robotique. Ce pont entre les deux domaines a été réalisé conjointement avec Nicolas Markey (maintenant à l'IRISA, Rennes), Nicolas Perrin (ISIR, Paris) et Philippe Schlehuber-Caissier (maintenant au LRDE, Paris).
http://www.lsv.fr/_~bouyer/

*Intervenant

Prix de Thèse du GDR GPL

Procédures de décision pour l'analyse de vulnérabilités

Benjamin Farinier * [1]

[1] CEA LIST (CEA-LIST) – Commissariat à l'énergie atomique et aux énergies alternatives – CEA LIST, LSL, Université Paris-Saclay, Gif-sur-Yvette, France

L'Exécution symbolique est une technique de vérification formelle qui consiste en modéliser les exécutions d'un programme par des formules logiques pour montrer que ces exécutions vérifient une propriété donnée. Très efficace pour la recherche de bogues, il est question aujourd'hui de l'employer dans d'autres contextes, comme en analyse de vulnérabilités. L'application de l'Exécution symbolique à l'analyse de vulnérabilités diffère de la recherche de bogues sur au moins deux aspects:

- Les formules logiques générées au cours de l'Exécution symbolique deviennent rapidement gigantesques et de plus en plus difficiles à résoudre pour les solveurs.

- La modélisation de certaines propriétés de sécurité est susceptible de faire intervenir des quantificateurs dont l'emploi rend les formules logiques générées presque impossibles à résoudre.

Cette thèse porte donc sur ces deux problématiques issues du domaine des procédures de décision, visant à permettre des modélisations plus fines nécessaires à l'analyse de vulnérabilités.

*Intervenant

EMI: Une approche pour unifier l'analyse et l'exécution embarquée à l'aide d'un interpréteur de modèles pilotable

Valentin Besnard * [1]

[1] ESEO-Tech – Université Bretagne Loire – 10 boulevard JeanneteauCS 9071749107 Angers cedex 2, France

La complexité croissante des systèmes embarqués les expose à davantage de bogues logiciels, d'erreurs de conception et de failles de sécurité. Les besoins en vérification et en validation sont donc de plus en plus importants.
Pour exécuter et analyser des modèles de ces systèmes, des transformations sont généralement nécessaires pour obtenir (i) le code exécutable pouvant être déployé sur une cible embarquée et (ii) des modèles d'analyse permettant d'appliquer des techniques de vérification formelle (p. ex. de model-checking). Cependant, ces transformations typiquement non-prouvées sont à l'origine de fossés sémantiques et nécessitent d'établir une relation d'équivalence entre le code exécutable et les modèles d'analyse afin de garantir que ce qui est exécuté est bien ce qui a été vérifié. Pour unifier les activités d'analyse et l'exécution embarquée de modèles, l'approche EMI repose sur un interpréteur de modèles pilotable permettant d'utiliser un unique couple (modèle + sémantique) pour toutes les activités de développement logiciel. Pour évaluer cette approche, un interpréteur de modèles UML a été conçu et appliqué à différents cas d'études de systèmes embarqués afin de mettre en œuvre diverses activités d'analyse (p. ex. simulation, animation, débogage, model-checking, monitoring).

*Intervenant

Sémantique Mécanisée et Compilation Vérifiée pour un Langage Synchrone à Flots de Données avec Réinitialisation

Lélio Brun * [1]

[1] Ecole Normale Supérieure Paris-Saclay (ENS Paris Saclay) – université Paris Dauphine, PSL Resarch University – 61 av du Pdt Wilson94230 Cachan, France

Les spécifications basées sur les schémas-blocs et machines à états sont utilisées pour la conception de systèmes de contrôle-commande, particulièrement dans le développement d'applications critiques. Des outils tels que Scade et Simulink/Stateflow sont équipés de compilateurs qui traduisent de telles spécifications en code exécutable. Ils proposent des langages de programmation permettant de composer des fonctions sur des flots, tel que l'illustre le langage synchrone à flots de données Lustre.

Cette thèse présente Vélus, un compilateur Lustre vérifié dans l'assistant de preuves interactif Coq. Nous développons des modèles sémantiques pour les langages de la chaîne de compilation, et utilisons le compilateur C vérifié CompCert pour générer du code exécutable et donner une preuve de correction de bout en bout. Le défi principal est de montrer la préservation de la sémantique entre le paradigme flots de données et le paradigme impératif, et de raisonner sur la représentation bas niveau de l'état d'un programme.

En particulier, nous traitons le reset modulaire, une primitive pour réinitialiser des sous-systèmes. Ceci implique la mise en place de modèles sémantiques adéquats, d'algorithmes de compilation et des preuves de correction correspondantes. Nous présentons un nouveau langage intermédiaire dans le schéma habituel de compilation modulaire dirigé par les horloges de Lustre. Ceci débouche sur l'implémentation de passes de compilation permettant de générer un meilleur code séquentiel, et facilite le raisonnement sur la correction des transformations successives du reset modulaire.

*Intervenant

Atelier "Préparer votre dossier de Candidature"

Atelier pour les jeunes chercheurs de préparation aux concours CNRS

Sandrine Blazy * [1]

[1] Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA) – Universite de Rennes 1, Institut National des Sciences Appliquées - Rennes, Institut National des Sciences Appliquées, Université de Bretagne Sud, École normale supérieure - Rennes, Institut National de Recherche en Informatique et en Automatique, CentraleSupélec : UMR6074, Centre National de la Recherche Scientifique, IMT Atlantique Bretagne-Pays de la Loire, Institut Mines-Télécom [Paris] – Avenue du général LeclercCampus de Beaulieu 35042 RENNES CEDEX, France

Programme de l'atelier
- présentation du comité national par Sandrine Blazy et Laurence Duchien

- présentation par Jean-Louis Giavitto (directeur de recherche à l'IRCAM) du métier et de la carrière d'un chercheur au CNRS
- table ronde réunissant 3 chercheurs CR CNRS récemment recrutés, Thomas Degueule, Jacques-Henri Jourdan et Djamel Eddine Khelladi

*Intervenant

AFADL
Sessions communes
avec MTV2 et LVP

Un prouveur automatique pour la géométrie projective et son intégration à Coq

Nicolas Magaud * [1]

[1] Laboratoire des sciences de l'ingénieur, de l'informatique et de l'imagerie (ICube) – Centre National de la Recherche Scientifique : UMR7357, université de Strasbourg : FR3627 – 300 bd Sébastien Brant - BP 10413 - F-67412 Illkirch Cedex, France

Afin de pouvoir démontrer formellement la correction d'algorithmes géométriques en Coq, il est nécessaire de disposer d'outils pour automatiser au moins partiellement les démonstrations en géométrie. Nous étudions cette question dans le cadre simple de la géométrie projective en utilisant une approche combinatoire et la notion de rang d'un ensemble de points. L'outil proposé, implanté en C, procède par saturation du contexte et permet de démontrer automatiquement de nombreux théorèmes emblématiques de la géométrie projective. Afin de s'assurer de la correction de ces démonstrations, l'outil produit une trace sous la forme d'un script de preuve, qui est ensuite vérifié par Coq.

*Intervenant

Unsolvability of the Quintic Formalized in Dependent Type Theory

Sophie Bernard [1], Cyril Cohen * [2], Assia Mahboubi [3,4], Pierre-Yves Strub [5]

[1] Université Côte d'Azur – INRIA – France
[2] Université Côte d'Azur – INRIA – France
[3] Gallinette Project Team – INRIA – France
[4] Vrije Universiteit, Amsterdam – Pays-Bas
[5] LIX – Ecole Polytechnique – France

In this talk, we describe an axiom-free Coq formalization that there does not exists a general method for solving by radicals polynomial equations of degree greater than 4. This development includes a proof of Galois' Theorem of the equivalence between solvable extensions and extensions solvable by radicals. The unsolvability of the general quintic follows from applying this theorem to a well chosen polynomial with unsolvable Galois group.

*Intervenant

Modèle de fuite structurée & applications à la compilation sécurisée

Gilles Barthe [1,2], Benjamin Gregoire [3], Vincent Laporte * [4], Swarn Priya [3]

[1] MPI-SP – Allemagne
[2] IMDEA Software Institute – Espagne
[3] Inria Sophia Antipolis – Université Côte d'Azur, Inria – France
[4] LORIA – CNRS, Inria, Université de Lorraine – F-54500 Nancy, France

Un compilateur correct permet d'étudier les comportements du programme cible - résultat de la compilation - en observant uniquement le programme source : nul besoin d'observer le programme généré ni de comprendre l'implémentation du compilateur pour savoir que le programme cible calcule la même fonction que le programme source correspondant.

Cependant, cette faculté est limitée aux propriétés fonctionnelles des comportements du programme cible : on sait ce que le programme cible calcule - la même chose que le programme source - mais on ne sait pas comment. Pourtant, connaître des propriétés non fonctionnelles du programme cible est souvent instructif : quel est le coût (p.ex. : le temps) d'une exécution du programme ? le programme fait-il fuir des données sensibles par des canaux auxiliaires ? etc.

La sémantique des langages de programmation peut être étendue (on dit aussi " instrumentée ") et rendue plus précise pour permettre de décrire de telles propriétés (ou hyper-propriétés) non-fonctionnelles ; cependant, la correction des compilateurs ne peut pas, en général, être étendue à ces sémantiques instrumentées : les optimisations peuvent modifier les effets non-fonctionnels rendus visibles par l'instrumentation (que l'on nomme la " fuite " d'une exécution).

Dans ce travail nous présentons un modèle de fuite structurée comme un moyen d'instrumenter les sémantiques des langages de programmation pour permettre de décrire un éventail de propriétés (et hyper-propriétés) non-fonctionnelles d'intérêt comme le coût à l'exécution ou la politique de sécurité " constant-time ". Étant entendu qu'un compilateur ne préserve pas, en général, la fuite associée à l'exécution d'un programme, le caractère structuré de cette fuite permet toutefois de décrire simplement l'effet de la compilation sur celle-ci. Plus précisément, en plus d'un programme cible, le compilateur produit un " transformateur de fuites " qui décrit comment calculer la fuite d'une exécution cible à partir de la fuite de l'exécution correspondante dans le programme source. Ces transformateurs de fuite sont exprimés dans un langage dédié, ce qui permet de les employer pour justifier la préservation de propriétés non-fonctionnelles mais aussi pour décrire comment ces propriétés sont modifiées.

L'application de cette méthodologie au compilateur Jasmin (dont la correction est formellement vérifiée en Coq) a ainsi permis de prouver que ce compilateur préserve la contre-mesure dite " constant-time " et de transférer au niveau assembleur des résultats d'une analyse de coût

*Intervenant

effectuée au niveau source. Cette analyse peut donc s'appuyer sur les abstractions et structures de haut niveau qui sont explicites au niveau source et ainsi être bien plus précise qu'une analyse similaire effectuée directement sur un programme assembleur.

Binsec/Rel : Exécution Symbolique Relationnelle Efficace pour Analyse de Binaire Constant-Time

Lesly-Ann Daniel * [1], Sébastien Bardin [1], Tamara Rezk [2]

[1] CEA LIST – Université Paris Saclay – France
[2] Inria Sophia Antipolis – Institut National de Recherche en Informatique et en Automatique – 2004 route des Lucioles BP 93 06902 Sophia Antipolis, France

Constant-time est une contre-mesure aux attaques temporelles qui interdit les branchements et les accès mémoires dépendants des secrets. Cette contre-mesure n'est généralement pas préservée par le compilateur et requiert donc de raisonner au niveau binaire. Or, les outils d'analyse dédiés à constant-time raisonnent actuellement à un plus haut niveau (C ou LLVM), approximent la sémantique du programme, ou ne passent pas à l'échelle. Nous concevons une technique d'analyse efficace au niveau binaire qui ne fait pas d'approximation sur la sémantique du programme, permettant à la fois de trouver des bugs ou de faire de la vérification bornée pour constant-time. Celle-ci s'appuie sur l'exécution symbolique relationnelle, à laquelle nous ajoutons des optimisations dédiées. Nous proposons un prototype, Binsec/Rel et réalisons des expériences sur un ensemble de 338 binaires cryptographiques, démontrant le passage à l'échelle de notre technique. De plus, en utilisant Binsec/Rel, nous avons automatisé et étendu une étude existante sur la préservation de constant-time par les compilateurs. Nous avons ainsi découvert des violations introduites par les compilateurs qui étaient hors de portée des outils d'analyse pour LLVM, soulignant l'importance de raisonner au niveau binaire.

*Intervenant

Binary-level Directed Fuzzing for Use-After-Free Vulnerabilities

Manh-Dung Nguyen * [1]

[1] CEA LIST – Université Paris Saclay – France

Directed fuzzing focuses on automatically testing specific parts of the code by taking advantage of additional information such as (partial) bug stack trace, patches or risky operations. Key applications include bug reproduction, patch testing and static analysis report verification. Although directed fuzzing has received a lot of attention recently, hard-to-detect vulnerabilities such as Use-Afer-Free (UAF) are still not well addressed, more especially at the binary level. We propose UAFuzz, the first (binary-level) directed greybox fuzzer dedicated to UAF bugs. The technique features a fuzzing engine tailored to UAF specifics, a lightweight code instrumentation and an efficient bug triage step. Experimental evaluation for bug reproduction on real cases demonstrates that UAFuzz significantly outperforms state-of-the-art directed fuzzers in terms of fault detection rate, time to exposure and bug triaging. UAFuzz has also been proven effective in patch testing, leading to the discovery of 20 new bugs in Perl, GPAC and GNU Patch (including a buggy patch) - all of them have been acknowledged and 14 have been fixed. Last but not least, we provide to the community the first fuzzing benchmark dedicated to UAF, built on both real codes and real bugs.

*Intervenant

Détection d'objectifs de test polluants pour les critères de flot de données

Thibault Martin * [1], Kosmatov Nikolai [1,2], Virgile Prevosto [1], Matthieu Lemerre [1]

[1] CEA LIST – Université Paris Saclay – France
[2] Thales Research and Technology [Palaiseau] – THALES – 1 Avenue Augustin Fresnel, 91767 Palaiseau cedex, France

Dans ce travail, nous évaluons trois approches pour détecter les objectifs de test polluants pour les critères de flot de données, avec une analyse de flot de données simple, une analyse de valeurs, basée sur l'interprétation abstraite, et une analyse de plus faible précondition. Nous avons implanté ces approches dans LT EST, une boîte à outils open-source pour le test. Nous avons ensuite évalué et comparé ces techniques sur différentes études de cas et avons analysé leurs capacités à détecter les objectifs polluants et leurs limites. Nous nous sommes concentrés sur la partie clef des critères de flot de données : les paires def-use. Les capacités de détection que nous avons observées semblent être différentes de celles des expériences similaires faites précédemment pour d'autres critères.

Cette soumission est un résumé long du papier "Detection of Polluting Test Objectives for Dataflow Criteria" paru à IFM 2020.

*Intervenant

Identifying and Generating Missing Tests using Machine Learning on Execution Traces

Frédéric Tamagnan * [1,2]

[1] Franche-Comté Électronique Mécanique, Thermique et Optique - Sciences et Technologies (UMR 6174) (FEMTO-ST) – Université de Technologie de Belfort-Montbeliard, Ecole Nationale Supérieure de Mécanique et des Microtechniques, Centre National de la Recherche Scientifique : UMR6174, Université de Franche-Comté, Université Bourgogne Franche-Comté [COMUE] – 32 avenue de lÓbservatoire 25044 BESANCON CEDEX, France

[2] Orange – Orange Group – France

Testing IT systems has become a major bottleneck for many companies. Besides the growing complexity of such systems, shorter release cycles and increasing quality requirements have led to increased verification and validation costs. However, analysis of existing testing procedures reveals that not all artifacts are exploited to tame this cost increase. In particular, customer traces are usually ignored by validation engineers. In this paper, we use machine learning from execution traces (both customer traces and test execution traces) to identify test needs and to generate new tests in the context of web services and API testing. Log files of customer traces are split into smaller traces (user sessions) then encoded into Pandas DataFrames for data analysis and machine learning. Clustering algorithms are used to analyse the customer traces and compare them with existing system tests, and machine learning models are used to generate missing tests in the desired clusters. The tool-set is implemented in an open-source library called Agilkia.

*Intervenant

Pas de Pannes, Pas d'Exploits: Vérification Automatique de Noyaux Embarqués

Olivier Nicole [1], Matthieu Lemerre [1], Sébastien Bardin * [1], Xavier Rival [2]

[1] CEA LIST – Université Paris Saclay, Université Paris-Saclay – France
[2] Département d'informatique de l'ENS – Ecole Normale Supérieure de Paris - ENS Paris, CNRS : UMR8548, PSL University, Inria Paris – France

Rejeu de la publication suivante acceptée à RTAS 2021 :
No Crash, No Exploit: Automated Verification of Embedded Kernels
The kernel is the most safety- and security-critical component of many computer systems, as the most severe bugs lead to complete system crash or exploit. It is thus desirable to guarantee that a kernel is free from these bugs using formal methods, but the high cost and expertise required to do so are deterrent to wide applicability. We propose a method that can verify both absence of runtime errors (i.e. crashes) and absence of privilege escalation (i.e. exploits) in embedded kernels from their binary executables. The method can verify the kernel runtime independently from the application, at the expense of only a few lines of simple annotations. When given a specific application, the method can verify simple kernels without any human intervention. We demonstrate our method on two different use cases: we use our tool to help the development of a new embedded real-time kernel, and we verify an existing industrial real-time kernel executable with no modification. Results show that the method is fast, simple to use, and can prevent real errors and security vulnerabilities.

*Intervenant

Session GT CLAP

Sound Semantic Static Analysis for Multiple Languages in the MOPSA Project

Antoine Miné * [1]

[1] Laboratoire dÍnformatique de Paris 6 (LIP6) – Sorbonne Universite, Centre National de la Recherche Scientifique : UMR7606 – 4 Place JUSSIEU 75252 PARIS CEDEX 05, France

We present MOPSA, an ongoing project to design a static analyzer by abstract interpretation targeting multiple languages. Static analyzers aim at inferring, at compile time, properties of program executions, and help ensuring their correctness. They perform an interpretation of the program in an abstract domain of properties that is approximate (to ensure efficiency) but sound with respect to the semantic of the language (no false negative). The classic approach to multi-language analyzers is to use language-specific front-ends to translate programs into a common, lower-level language (e.g., a subset of C, an intermediate representation, or a bytecode) before the analysis. The back-end then iterates over a fixed language using a data abstraction which is composed of a collection of abstraction modules that can be plugged in and out to achieve various cost/precision trade-offs.
MOPSA strives to achieve a higher degree of modularity and extensibility by considering value abstractions, iterators, and control-flow abstractions uniformly as domain modules. Each module can extend the language syntax and rewrite expressions and statements to simpler ones dynamically to be processed by further modules. Hence, we avoid the pitfalls of fixing a common intermediate representation and we allow the rewriting to exploit any information found by the analysis so far. We will present MOPSA's architecture and some of our applications. These include a value analysis for C programs that also uses a specification language for C library functions. We also present type and value analyses for a significant subset of Python, as well as a value analysis for Python programs calling native C functions. Despite the variation in the languages analyzed and the properties inferred, these analyses share many common abstractions.

*Intervenant

Verification of FIFO systems

Etienne Lozes * [1]

[1] Laboratoire dÍnformatique, Signaux, et Systèmes de Sophia Antipolis (I3S) – Université Nice Sophia Antipolis (... - 2019), COMUE Université Côte dÁzur (2015 - 2019), Centre National de la Recherche Scientifique : UMR7271, Université Côte d'Azur, COMUE Université Côte dÁzur (2015 - 2019), COMUE Université Côte dÁzur (2015 - 2019) – 2000, route des Lucioles - Les Algorithmes - bât. Euclide B 06900 Sophia Antipolis, France

FIFO systems are systems of automata communicating through FIFO queues. This simple and rather idealised model can be used to analyse some aspects of message-passing systems, reactive systems with event queues, weak memory models with buffered reads and writes, etc. From
a purely computational perspective, this is a Turing complete model even for just one automaton and one FIFO queue.
In this talk, I will present a personal selection of existing works on the problem of the automatic (push-button) verification of such systems. I will consider in particular the works that try to address the verification of FIFO systems that are "nearly" systems with rendez-vous synchronisation.
Nearly synchronous FIFO systems can be found for instance in the work of Lipton on reduction [1], Elrad&Francez on communication closed layers [2],
Bultan et al on synchronisability [3], Mushcoll et al on existential boundedness [4], or more recently Bouajjani et al on k-synchronous systems [5].
This idea is also implicitly present in several works on multi-party session types [6], although the connection there is only well understood in the bipartite setting, where it matches the notion of half-duplex communications [7,8]. The aim of the talk will also be to present recent personal contributions and ongoing works on k-synchronous systems, existentially-bounded systems, and half-duplex systems [9,10,11,12].

Lipton, R.J.: Reduction: A method of proving properties of parallel programs. Commun. ACM 18(12), 717–721 (1975).https://doi.org/10.1145/361227.361234

Elrad, T., Francez, N.: Decomposition of distributed programs into communication-closed layers. Sci. Comput. Program. 2(3), 155–173 (1982).https://doi.org/10.1016/0167-6423(83)90013-8

Basu, S., Bultan, T.: On deciding synchronizability for asynchronously communicating systems. Theor. Comput. Sci.https://doi.org/10.1016/j.tcs.2016.09.023

*Intervenant

Model-Bounded Monitoring of Hybrid Systems

Etienne André * [1]

[1] Université de Lorraine, CNRS, Inria, LORIA, Nancy, France – Université de Lorraine – France

Monitoring of hybrid systems attracts both scientific and practical attention. However, monitoring algorithms suffer from the methodological difficulty of only observing sampled discrete-time signals, while real behaviors are continuous-time signals. To mitigate this problem of sampling uncertainties, we introduce a model-bounded monitoring scheme, where we use prior knowledge about the target system to prune interpolation candidates. Technically, we express such prior knowledge by linear hybrid automata (LHAs)-the LHAs are called bounding models. We introduce a novel notion of monitored language of LHAs, and we reduce the monitoring problem to the membership problem of the monitored language. We present two partial algorithms-one is via reduction to reachability in LHAs and the other is a direct one using polyhedra-and show that these methods, and thus the proposed model-bounded monitoring scheme, are efficient and practically relevant. This is a joint work with Masaki Waga and Ichiro Hasuo.

*Intervenant

Session GT IE

Une revue de littérature sur les recommandations pour la création de programmes d'éducation, d'entrainement et de sensibilisation à la sécurité informatique

Florence Sèdes [1], Olivier De Casanove *

[1] Institut de recherche en informatique de Toulouse (IRIT) – Université Toulouse I [UT1] Capitole, Université des Sciences Sociales - Toulouse I, Institut National Polytechnique de Toulouse - INPT, Université Paul Sabatier [UPS] - Toulouse III, CNRS : UMR5505, Université Toulouse le Mirail - Toulouse II, Université Toulouse I (UT1) Capitole, Université Paul Sabatier (UPS) - Toulouse III – 118 Route de Narbonne, F-31062 Toulouse Cedex 9, France

En s'appuyant sur des normes et des standards de sécurité informatique, il est possible d'établir des politiques de sécurité. Cependant, ces normes et standards sont très descriptifs, particulièrement lorsqu'ils portent sur la prévention et la sensibilisation. Or, ces deux aspects sont des enjeux majeurs bien en amont de la mise en ouevre de la sécurité informatique "moderne".
Pour répondre à ces enjeux, nous pouvons utiliser les programmes d'education, d'entraînement, de sensibilisation à la sécurité informatique (en anglais "SETA programs" (Security Education Training and Awareness)). Ces programmes permettent de trouver un équilibre stratégique entre le paradigme de prévention et de protection.
Nous proposons d'étudier l'état de l'art pour identifier des recommandations sur comment concevoir des programmes SETA. Les recommandations suivront le cycle PDCA (Plan Do Check Action) pour être les plus prescriptives possibles.

*Intervenant

Un survey sur les méthodes formelles en IE

Sophie Ebersold * [1]

[1] Institut de recherche en informatique de Toulouse (IRIT) – université Toulouse 1 Capitole, Université Fédérale Toulouse Midi-Pyrénées, Université Toulouse - Jean Jaurès, Université Toulouse III - Paul Sabatier, Université Fédérale Toulouse Midi-Pyrénées : UMR5505, Centre National de la Recherche Scientifique, Institut National Polytechnique (Toulouse) – 118 Route de Narbonne, F-31062 Toulouse Cedex 9, France

L'étude que nous avons publiée dans ACM Computing Surveys examine certaines des principales approches formelles et les compare aux méthodes informelles.
L'analyse que nous avons menée utilise un ensemble de 9 critères complémentaires, tels que le niveau d'abstraction, la disponibilité des outils, le support de la traçabilité. Elle est mise en oeuvre sur 22 approches différentes, classées en cinq catégories : polyvalentes, langage naturel, graphes/automates, autres notations mathématiques, sans couture (basée sur un langage de programmation). Ce survey aborde également un certain nombre de questions ouvertes, notamment le rôle des outils et de l'éducation, et la manière de faire profiter davantage les applications industrielles des contributions des approches formelles.

*Intervenant

Panorama sur les exigences de sécurité

Ludovic Apvrille * [1]

[1] Telecom Paris – Télécom ParisTech – France

La présentation expliquera tout d'abord comment les exigences de sécurité se situent dans le cycle de développement d'un système. Nous ferons ensuite une revue exhaustive des différents types d'exigences de sécurité, en contextualisant ces types d'exigences sous la forme d'expressions simples et textuelles. Par la suite, nous montrerons, pour certaines de ces exigences, comment elles peuvent être exprimées au regard de diagrammes UML / SysML, et comment elles peuvent être formellement exprimées et vérifiées avec l'outil TTool.

*Intervenant

Session GT VL

An approach and benchmark to detect behavioral changes of commits in continuous integration

Benjamin Danglot * [1]

[1] Davidson – Davidson College – France

When a developer pushes a change to an application's codebase, a good practice is to have a test case specifying this behavioral change. Thanks to continuous integration (CI), the test is run on subsequent commits to check that they do no introduce a regression for that behavior. In this paper, we propose an approach that detects behavioral changes in commits. As input, it takes a program, its test suite, and a commit. Its output is a set of test methods that capture the behavioral difference between the pre-commit and post-commit versions of the program. We call our approach DCI (Detecting behavioral changes in CI). It works by generating variations of the existing test cases through (i) assertion amplification and (ii) a search-based exploration of the input space. We evaluate our approach on a curated set of 60 commits from 6 open source Java projects. To our knowledge, this is the first ever curated dataset of real-world behavioral changes. Our evaluation shows that DCI is able to generate test methods that detect behavioral changes. Our approach is fully automated and can be integrated into current development processes. The main limitations are that it targets unit tests and works on a relatively small fraction of commits. More specifically, DCI works on commits that have a unit test that already executes the modified code. In practice, from our benchmark projects, we found 15.29% of commits to meet the conditions required by DCI.

*Intervenant

Deep Software Variability: Towards Handling Cross-Layer Configuration

Luc Lesoil * [1]

[1] Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA) – Universite de Rennes 1 : UMR6074, Université de Rennes, Institut National des Sciences Appliquées - Rennes, Institut National des Sciences Appliquées, Université de Bretagne Sud, École normale supérieure - Rennes, Institut National de Recherche en Informatique et en Automatique, CentraleSupélec, Centre National de la Recherche Scientifique, IMT Atlantique Bretagne-Pays de la Loire, Institut Mines-Télécom [Paris] – Avenue du général LeclercCampus de Beaulieu 35042 RENNES CEDEX, France

Configuring software is a powerful means to reach functional and performance goals of a system. However, many layers (hardware, operating system, input data, etc.), themselves subject to variability, can alter performances of software configurations. For instance, configurations' options of the x264 video encoder may have very different effects on x264's encoding time when used with different input videos, depending on the hardware on which it is executed. In this vision paper, we coin the term deep software variability to refer to the interaction of all external layers modifying the behavior or non-functional properties of a software. Deep software variability challenges practitioners and researchers: the combinatorial explosion of possible executing environments complicates the understanding, the configuration, the maintenance, the debug, and the test of configurable systems. There are also opportunities: harnessing all variability layers (and not only the software layer) can lead to more efficient systems and configuration knowledge that truly generalizes to any usage and context.

*Intervenant

Expanding the Number of Reviewers in Open-Source Projects by Recommending Appropriate Developers

Aleksandr Chueshev * [1]

[1] LIP6 – Sorbonne Universite : UMR7606, Centre National de la Recherche Scientifique – 4 Place JUSSIEU 75252 PARIS CEDEX 05, France

Code review is an important part of the development of any software project. Recently, many open source projects have begun practicing lightweight and tool-based code review (a.k.a modern code review) to make the process simpler and more efficient. However, those practices still require reviewers, of which there may not be sufficiently many to ensure timely decisions. In this paper, we propose a recommender-based approach to be used by open-source projects to increase the number of reviewers from among the appropriate developers. We first motivate our approach by an exploratory study of nine projects hosted on GitHub and Gerrit. Secondly, we build the recommender system itself, which, given a code change, initially searches for relevant reviewers based on similarities between the reviewing history and the files affected by the change, and then augments this set with developers who have a similar development history as these reviewers but have little or no relevant reviewing experience. To make these recommendations, we rely on collaborative filtering, and more precisely, on matrix factorization. Our evaluation shows that all nine projects could benefit from our system by using it both to get recommendations of previous reviewers and to expand their number from among the appropriate developers.

*Intervenant

Empirical Study of Restarted and Flaky Builds on Travis CI

Thomas Durieux * [1]

[1] CASTOR | CASTOR - Software Research Centre - KTH – KTH Royal Institute of TechnologySE-100 44 StockholmSweden, Suède

Continuous Integration (CI) is a development practice where developers frequently integrate code into a common codebase. After the code is integrated, the CI server runs a test suite and other tools to produce a set of reports (e.g., the output of linters and tests). If the result of a CI test run is unexpected, developers have the option to manually restart the build, re-running the same test suite on the same code; this can reveal build flakiness, if the restarted build outcome differs from the original build. In this study, we analyze restarted builds, flaky builds, and their impact on the development workflow. We observe that developers restart at least 1.72% of builds, amounting to 56,522 restarted builds in our Travis CI dataset. We observe that more mature and more complex projects are more likely to include restarted builds. The restarted builds are mostly builds that are initially failing due to a test, network problem, or a Travis CI limitations such as execution timeout. Finally, we observe that restarted builds have an impact on development workflow. Indeed, in 54.42% of the restarted builds, the developers analyze and restart a build within an hour of the initial build execution. This suggests that developers wait for CI results, interrupting their workflow to address the issue. Restarted builds also slow down the merging of pull requests by a factor of three, bringing median merging time from 16h to 48h.

*Intervenant

Session GT GLIA

Bug or not bug? That is the question.

Quentin Perez * [1], Pierre-Antoine Jean [1], Christelle Urtado [1], Sylvain Vauttier [1]

[1] EuroMov - Digital Health in Motion (Euromov DHM) – IMT - MINES ALES, Institut Mines-Télécom [Paris], Université de Montpellier : $UR_U M_I MT_1 02$ – $-UniversitdeMontpellierUFRSTAPS700avenueduPicSaintLoup34090Montpellier, France$

Nowadays, development teams often rely on tools such as Jira or Bugzilla to manage backlogs of issues to be solved to develop or maintain software. Although they relate to many different concerns (e.g., bug fixing, new feature development, architecture refactoring), few means are proposed to identify and classify these different kinds of issues, except for non mandatory labels that can be manually associated to them. This may lead to a lack of issue classification or to issue misclassification that may impact automatic issue management (planning, assignment) or issue-derived metrics. Automatic issue classification thus is a relevant topic for assisting backlog management. This paper proposes a binary classification solution for discriminating bug from non bug issues. This solution combines natural language processing (TF-IDF) and classification (multi-layer perceptron) techniques, selected after comparing commonly used solutions to classify issues. Moreover, hyper-parameters of the neural network are optimized using a genetic algorithm. The obtained results, as compared to existing works on a commonly used benchmark, show significant improvements on the F1 measure for all datasets.

*Intervenant

Transfer learning of Kernel configuration performance across Linux versions

Hugo Martin * [1]

[1] Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA) – Universite de Rennes 1, Université de Rennes, Institut National des Sciences Appliquées - Rennes, Institut National des Sciences Appliquées, Université de Bretagne Sud, École normale supérieure - Rennes, Institut National de Recherche en Informatique et en Automatique, CentraleSupélec : UMR6074, Centre National de la Recherche Scientifique, IMT Atlantique Bretagne-Pays de la Loire, Institut Mines-Télécom [Paris] – Avenue du général LeclercCampus de Beaulieu 35042 RENNES CEDEX, France

With large scale and complex configurable systems, it is hard for users to choose the right combination of options (i.e., configurations) in order to obtain the wanted trade off between functionality and performance goals such as speed or size. Machine learning can help in relating these goals to the configurable system options, and thus, predict the effect of options on the outcome, typically after a costly training step. However, many configurable systems evolve at such a rapid pace that it is impractical to retrain a new model from scratch for each new version. In this paper, we propose a new method to enable transfer learning of performance predictions among versions of the same configurable system. Taking the extreme case of the Linux kernel with its 14.500 configuration options, we first investigate how performance predictions of kernel size degrade over successive versions. We show that the direct reuse of an accurate prediction model from 2017 quickly becomes inaccurate when Linux evolves, up to a 32% mean error by August 2020. We thus propose a new approach for transfer evolution-aware model shifting (tEAMS). It leverages the structure of a configurable system to transfer an initial predictive model towards its future versions with a minimal amount of extra processing for each version. We show that outperforms state of the art approaches and using tEAMS on the history of Linux kernels from 4.13 to 5.8, spanning over a period of 3 years, the model accuracy stays low and roughly constant over time.

*Intervenant

Planification automatique des reconfigurations des systèmes distribués

Simon Robillard * [1]

[1] IMT Atlantique Bretagne-Pays de la Loire (IMT Atlantique) – Ministère de l'Économie et de l'Industrie – Campus Brest : Technopôle Brest-Iroise CS 8381829238 BREST Cedex 3 -Campus Nantes : 4, rue Alfred Kastler- La chantrerie 44300 NANTES -Campus Rennes : 2 Rue de la Châtaigneraie, 35510 CESSON SEVIGNE, France

Les systèmes distribués à grande échelle jouent aujourd'hui un rôle important dans de nombreux domaines, où leur adaptabilité est un atout. Pourtant les reconfigurations de ces systèmes continuent le plus souvent d'être effectuées de manière ad hoc, un processus qui est à la fois inefficace et source potentielle d'erreurs. Cette présentation décrira une technique utilisant un solveur SMT pour résoudre le problème de la planification de reconfiguration d'un système distribué dans un modèle à composants. Plus précisément, étant donné un ensemble de tâches à exécuter et un état final désiré pour le système, nous montrerons comment générer un plan de reconfiguration qui satisfasse les dépendances entre composants et qui soit aussi optimisé pour l'exécution parallèle.

*Intervenant

Wise Object

Ilham Alloui * [1], Flavien Vernier *

1

[1] Laboratoire dÍnformatique, Systèmes, Traitement de lÍnformation et de la Connaissance (LISTIC) – Université Savoie Mont Blanc : EA3703 – BP 80439 74944 ANNECY LE VIEUX Cedex, France

L'essor de l'IA depuis ces 2 dernières décennies a entrainé dans son évolution tous les domaines scientifiques. Dans cette mouvance, nous nous intéressons à l'ingénierie logicielle des systèmes qui intègrent de l'intelligence. Notre approche est fondée sur la séparation des aspects métier, de l'IA et de la sémantique de l'utilisateur final du système. La première étape a été de travailler sur les mécanismes de base sur lesquels les systèmes intelligents pourront être construits : le monitoring, l'apprentissage, l'analyse, la prise de décision... Nous avons proposé dans ce contexte, le concept d'objet sage (wise object) comme la brique de base qui permet au système logiciel de construire une connaissance (sagesse) sur les services métier qu'il est censé fournir ainsi que sur la manière dont ces services sont utilisés. Un premier framework a été développé dans ce sens et fournit à des objets logiciels la capacité d'apprendre par eux-mêmes leur fonctionnement, ainsi que la manière habituelle dont ils sont utilisés. Cet exposé présentera de manière générale les concepts, l'implémentation du framework et les perspectives de ces travaux.

*Intervenant

Session GT Secu-GL

Identification automatique des vulnérabilités de sécurité dans les systèmes logiciels

Raounak Benabidallah * [1]

[1] Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA) – Universite de Rennes 1, Université de Rennes, Institut National des Sciences Appliquées - Rennes, Institut National des Sciences Appliquées, Université de Bretagne Sud, École normale supérieure - Rennes, Institut National de Recherche en Informatique et en Automatique, CentraleSupélec : UMR6074, Centre National de la Recherche Scientifique, IMT Atlantique Bretagne-Pays de la Loire, Institut Mines-Télécom [Paris] – Avenue du général LeclercCampus de Beaulieu 35042 RENNES CEDEX, France

La menace posée par les vulnérabilités logicielles croît de manière exponentielle. Ce phénomène est dû, d'une part, à l'omniprésence des logiciels, et d'autre part, au nombre important de failles existantes. Pour faire face à ce problème, plusieurs stratégies ont été élaborées au fil du temps. Certaines visent à mettre en place de bonnes pratiques de développement et les intégrer dès la phase de conception tandis que d'autres consistent à effectuer des inspections de sécurité en indiquant les zones vulnérables. Les travaux présentés s'inscrivent dans la deuxième catégorie et portent essentiellement sur la construction de modèles de prédiction de vulnérabilités. La création de ces derniers soulève différents problèmes. Le plus important étant le manque de données sur les vulnérabilités logicielles. À cet effet, nous mettons en place une chaîne de traitement complète allant de la création et l'annotation automatique d'un corpus de sécurité jusqu'à la construction et l'évaluation des modèles de prédiction de vulnérabilités. La première contribution est plus axée sur l'approche de construction de corpus que sur le corpus lui-même. L'approche est basée sur la conception de méta-scanners de vulnérabilités permettant d'identifier des vulnérabilités de code efficacement. Cela consiste à combiner plusieurs outils d'analyse statique en se basant sur leurs performances individuelles pour chaque catégorie de vulnérabilités. Notre deuxième contribution correspond au corpus SecureQualitas qui consiste en un corpus d'applications Java annotées avec les vulnérabilités qu'elles contiennent. Nous construisons ce corpus en utilisant un méta-scanner construit à l'aide de trois outils d'analyse de vulnérabilités. Enfin, notre troisième contribution est de construire un modèle de prédiction du code vulnérable. Nous avons opté pour l'utilisation de métriques de qualité pour caractériser le code et nous avons étudié les performances des modèles à la fois sur des catégories de vulnérabilités apprises par les modèles et sur des catégories non encore connues Les résultats de nos expérimentations ont montré l'efficacité des modèles sur les deux populations de vulnérabilités : connues et non connues.

*Intervenant

RAICC: Revealing Atypical Inter-Component Communication in Android Apps

Jordan Samhi * [1]

[1] Université du Luxembourg (TruX) – Luxembourg

La communication entre les composants (ICC) est un des mécanismes les plus importants dans les applications Android.
En effet, il permet aux développeurs de mettre en place de riches fonctionnalités et la réutilisation des composants entre les applications.
Bien que ce mécanisme rende la modélisation des applications Android difficile, des approches ont été développées (e.g., EPICC, ICCTA, AMANDROID, etc.) pour améliorer les analyses statiques d'applications Android en se concentrant sur les méthodes documentées (e.g., startActivity).
Dans ce travail nous montrons que les modèles existants sont incomplets car le framework Android possèdent d'autres méthodes atypiques pour effectuer de la communication entre les composants.
Nous proposons de récupérer systématiquement ces méthodes atypiques, ainsi qu'une approche statique, RAICC, pour la modélisation de nouveaux liens ICC pour booster les analyses existantes. Nous montrons que RAICC améliore la précision et le rappel des outils existants permettant de détecter des fuites de données dans des applications de benchmark. De plus, nous montrons empiriquement que les méthodes ICC atypiques sont largement utilisées dans les applications Android.
Enfin, nous montrons que RAICC augmente le nombre de liens ICC trouvés de 61,6 % dans des malware, et que RAICC permet la détection de potentielles nouvelles vulnérabilités ICC

*Intervenant

Présentation DU GT Secu-GL

Salah Sadou * [1]

[1] Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA) – Universite de Rennes 1, Université de Rennes, Institut National des Sciences Appliquées - Rennes, Institut National des Sciences Appliquées, Université de Bretagne Sud, École normale supérieure - Rennes, Institut National de Recherche en Informatique et en Automatique, CentraleSupélec : UMR6074, Centre National de la Recherche Scientifique, IMT Atlantique Bretagne-Pays de la Loire, Institut Mines-Télécom [Paris] – Avenue du général LeclercCampus de Beaulieu 35042 RENNES CEDEX, France

Dans cette partie nous ferons un tours de table pour évoquer la vision de la sécurité via le génie logiciel selon chacune des équipes participantes.

*Intervenant

Discussion sur l'organisation et les actions du GT Secu-GL

Salah Sadou * [1,2]

[1] Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA) – Universite de Rennes 1, Université de Rennes, Institut National des Sciences Appliquées - Rennes, Institut National des Sciences Appliquées, Université de Bretagne Sud, École normale supérieure - Rennes, Institut National de Recherche en Informatique et en Automatique, CentraleSupélec : UMR6074, Centre National de la Recherche Scientifique, IMT Atlantique Bretagne-Pays de la Loire, Institut Mines-Télécom [Paris] – Avenue du général LeclercCampus de Beaulieu 35042 RENNES CEDEX, France
[2] Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA) – Universite de Rennes 1, Université de Rennes, Institut National des Sciences Appliquées - Rennes, Institut National des Sciences Appliquées, Université de Bretagne Sud, École normale supérieure - Rennes, Institut National de Recherche en Informatique et en Automatique, CentraleSupélec : UMR6074, Centre National de la Recherche Scientifique, IMT Atlantique Bretagne-Pays de la Loire, Institut Mines-Télécom [Paris] – Avenue du général LeclercCampus de Beaulieu 35042 RENNES CEDEX, France

Nous discuterons ensemble sur la manière de fonctionner et les actions à entreprendre. Nous discuterons, aussi, de nos relations avec le GDR Sécurité.

*Intervenant

Session GT Debugging

Présentation du GT debugging

Steven Costiou * [1], Benoit Combemale *

[2]

[1] CRIStAL – Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France – France
[2] Diverse – Univ Rennes, CNRS, Inria, IRISA - UMR 6074 – France

Le GT debugging a pour objectif de rassembler dans une même communauté tout chercheur, ingénieur, équipe industrielle ou du GDR qui s'intéresse aux problèmes du debugging logiciel. Il s'inscrit dans la communauté du GDR Génie de la Programmation et du Logiciel (GPL) du CNRS. Durant cette présentation, nous introduiront le groupe, ses objectifs scientifiques et son organisation.

*Intervenant

Monilogging for Executable DSLs

Dorian Leroy * [1]

[1] Diverse – Univ Rennes, CNRS, Inria, IRISA - UMR 6074 – France

Les techniques dites de "runtime monitoring" et de "logging" sont centrales à l'analyse et à la supervision du comportement de programmes informatiques. Cependant, développer un support adéquat pour ces techniques entraîne des coûts importants qui peuvent décourager les ingénieurs de langages dédiés d'offrir ces capabilités aux utilisateurs de leur langage. De plus, ces deux techniques sont généralement considérées comme indépendantes et sont donc implémentées séparément, ce qui nuit à l'exploitation de leurs complémentarités. Pour répondre à ces problèmes, nous proposons MoniLog, un langage permettant de définir des loggers, des moniteurs, et des combinaisons des deux (i.e., des moniloggers), de façon agnostique au langage utilisé.

*Intervenant

Sub-method, partial behavioral reflection with Reflectivity

Steven Costiou [1], Vincent Aranega [2], Marcus Denker * [1]

[1] CRIStAL – Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France – France
[2] CRIStAL – Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France – France

This talk was given by Marcus Denker at the conference in March 2021 (https://2021.programming-conference.org/details/programming-2021-papers/5/Sub-method-partial-behavioral-reflection-with-Reflectivity-Looking-back-on-10-years)
Refining or altering existing behavior is the daily work of every developer, but that cannot be always anticipated, and software sometimes cannot be stopped. In such cases, unanticipated adaptation of running systems is of interest for many scenarios, ranging from functional upgrades to on-the-fly debugging or monitoring of critical applications.

A way of altering software at run time is using behavioral reflection, which is particularly well-suited for unanticipated adaptation of real-world systems. Partial behavioral reflection is not a new idea, and for years many efforts have been made to propose a practical way of expressing it. All these efforts resulted in practical solutions, but which introduced a semantic gap between the code that requires adaptation and the expression of the partial behavior. For example, in Aspect-Oriented Programming, a pointcut description is expressed in another language, which introduces a new distance between the behavior expression (the Advice) and the source code in itself.

Ten years ago, the idea of closing the gap between the code and the expression of the partial behavior led to the implementation of the Reflectivity framework. Using Reflectivity, developers annotate Ab- stract Syntax Tree (AST) nodes with meta-behavior which is taken into account by the compiler to produce behavioral variations. In this paper, we present Reflectivity, its API, its implementation and its usage in Pharo. We reflect on ten years of use of Reflectivity, and show how it has been used as a basic building block of many innovative ideas.

Reflectivity brings a practical way of working at the AST level, which is a high-level representation of the source code manipulated by software developers. It enables a powerful way of dynamically add and modify behavior. Reflectivity is also a flexible mean to bridge the gap between the expression of the meta-behavior and the source code. This ability to apply unanticipated adaptation and to provide behavioral reflection led to many experiments and projects during this last decade by external users. Existing work use Reflectivity to implement reflective libraries or languages extensions, featherweight code instrumentation, dynamic software update, debugging tools and visualization and software analysis tools.

*Intervenant

Reflectivity is actively used in research projects. During the past ten years, it served as a support, either for implementation or as a fundamental base, for many research work including PhD theses, conference, journal and workshop papers. Reflectivity is now an important library of the Pharo language, and is integrated at the heart of the platform.
Reflectivity exposes powerful abstractions to deal with partial behavioral adaptation, while providing a mature framework for unanticipated, non-intrusive and partial behavioral reflection based on AST annotation. Furthermore, even if Reflectivity found its home inside Pharo, it is not a pure Smalltalk-oriented solution. As validation over the practical use of Reflectivity in dynamic object-oriented languages, the API has been ported to Python. Finally, the AST annotation feature of Reflectivity opens new experimentation opportunities about the control that developers could gain on the behavior of their own software.

GT debugging: table ronde

Steven Costiou * [1], Benoit Combemale *

[2]

[1] CRIStAL – Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France – France
[2] Diverse – Univ Rennes, CNRS, Inria, IRISA - UMR 6074 – France

Lors de cette table ronde, nous échangerons sur les thèmes suivants :
- pour les nouveaux participants, qu'attendez-vous d'un GT sur le debugging ?

- quels défis souhaitez vous voir aborder ?

- comment ouvrir les problématiques du debugging aux autres GT ? Que pouvons nous leur apporter ? De quoi le debugging a besoin venant d'autres domaines ou thématiques ?

- quelles actions pour la suite du GT ?
- discussion ouverte

*Intervenant

Session GT Yoda

Online Testing of Dynamic Reconfigurations w.r.t. Adaptation Policies

Frédéric Dadeau [1], Jean-Philippe Gros [1], Olga Kouchnarenko * [1]

[1] FEMTO-ST Institute (FEMTO-ST) – Université de Bourgogne-Franche-Comté, CNRS : UMR6174 – 16 route de Gray, Besançon, France

Component systems are designed as sets of components that may reconfigure themselves according to adaptation policies. Adaptation policies are seen as artifacts that describe desirable behavior of the system without enforcing a specific one. An adaptation policy is designed as a set of rules that indicate, for a given set of configurations, which reconfiguration operations can be triggered, with fuzzy values representing their respective utility.
In this context, this presentation will present a model-based testing approach which aims to generate large test suites in order to measure the occurrences of reconfigurations and compare them to their utility values specified in the adaptation rules. This process is based on a usage model of the system used to stimulate the system and provoke reconfigurations. As the system may reconfigure dynamically, this online test generator observes the system responses and evolution in order to decide the next appropriate test step to perform. In the end, the relative frequencies of the reconfigurations are measured in order to determine if the adaptation policy is faithfully implemented. The approach is illustrated by simulations for platoons of (semi-)autonomous vehicles.

Note : Exposé en français avec un support en anglais

*Intervenant

On Reducing the Energy Consumption of Software Product Lines

Édouard Guegain * [1], Clément Quinton *

[1], Romain Rouvoy [1]

[1] Self-adaptation for distributed services and large software systems (SPIRALS) – Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL – F-59000 Lille, France

Over the last decade, several studies considered green software design as a key development concern to improve the energy efficiency of software. Yet, few techniques address this concern for Software Product Lines (SPL). In this paper, we, therefore, introduce two approaches to measure and reduce the energy consumption of an SPL by analyzing a limited set of products sampled from this SPL. While the first approach relies on the analysis of individual feature consumptions, the second one takes feature interactions into account to better mitigate the energy consumption of resulting products.

Our experimental results on a real-world SPL indicate that both approaches succeed to produce significant energy improvements on a large number of products, while consumption data were modelled from a small set of sampled products. Furthermore, we show that taking feature interactions into account leads to more products improved with higher energy savings per product.

*Intervenant

Presentation of GT Yoda

Rabéa Ameur-Boulifa * [1], Simon Bliudze *

[2], Hélène Coullon *

3

[1] Télécom Paris – Institut Polytechnique de Paris – 19 Place Marguerite Perey 91120 Palaiseau, France
[2] Self-adaptation for distributed services and large software systems (SPIRALS) – Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL – F-59000 Lille, France
[3] STACK - Software Stack for Massively Geo-Distributed Infrastructures – IMT Atlantique, Inria, LS2N, UBL, F-44307 Nantes, France – France

In recent years, distributed software systems have faced a set of new challenges raised by new Internet-scale distributed systems and highly dynamic infrastructures. Indeed, with the growth of the Internet-of-Things and Cyber-Physical Systems domains, new kinds of highly dynamic applications such as applications for smart-cities or Industry 4.0 have emerged, some of them being critical for cost or safety reasons thus calling for formal guarantees of correctness. Furthermore, new kinds of distributed utility computing paradigms have also recently and quickly entered the landscape such as Fog-, Edge- or mobile-computing, where devices may dynamically enter or leave the infrastructure. This rapid growth in dynamicity calls for programming support mature enough to provide safe and reliable adaptation mechanisms, but also such software-engineering-related properties as modularity, abstraction, and composability. This working group is intended to host discussions of new trends and foster contributions of the French community to the definition of adaptation mechanisms that would allow developers to design trustworthy and optimal dynamic distributed software and systems.
This group is open to a broad family of researchers from various communities: software engineering, languages, control theory, application domains (IoT, CPS), etc.

*Intervenant

CoSim20: un environnement pour la simulation collaborative et distribuée

Julien Deantoni * [1]

[1] Laboratoire dÍnformatique, Signaux, et Systèmes de Sophia Antipolis (I3S) – CNRS : UMR7271, Université Côte d'Azur (UCA), Polytech Nice-Sophia, Inria Sophia Antipolis - Méditerranée, GEMOC Initiative – 2004 rte des Lucioles (Lagrange L-041), BP93, F-06902 Sophia Antipolis Cedex, France

Les systèmes cyber-physiques sont des systèmes d'ingénierie complexe dans lesquels les parties computationnelles communiquent entre elles et avec les parties physiques décrivant l'environnement. Pour apprivoiser la complexité croissante de ces systèmes, ils sont généralement décomposés en différentes parties qui sont modélisées par différents experts, éventuellement issus de différentes organisations. Ces experts utilisent un langage adapté à leur problème, à la fois syntaxiquement et sémantiquement. À ce stade, d'une part on obtient des modèles exécutables des parties computationnelles et des modèles exécutables des parties physiques et, d'autre part, les modèles exécutables ne doivent pas violer les propriétés intellectuelles lorsqu'ils sont partagés et sont donc généralement partagés en tant qu'unités de simulation en boîte noire. Cependant, pour comprendre le comportement émergeant de l'ensemble du système, il est nécessaire de faire une simulation collaborative ; où les unités de simulation de différentes disciplines sont coordonnées pour échanger des données aux moments opportuns.

Les problèmes avec les approches existantes sont multiples. Premièrement, les unités de simulation exposées sous forme de boîtes noires ne permettent pas de prendre en compte les spécificités sémantiques de chaque modèle. Deuxièmement, les co-simulations sont aujourd'hui principalement basées sur une interface de programmation dirigée par le temps, et il a été montré que de telles interfaces introduisaient, lorsqu'elles sont appliquées à des systèmes cyber-physiques, des "retards artificiels" dus à la co-simulation elle-même. De tels retards impliquent une mauvaise précision qui peut invalider les résultats de la co-simulation. De plus, réduire ces délais de co-simulation implique de mauvaises performances globales de co-simulation. Troisièmement, la définition d'un cadre de co-simulation précis et performant peut être complexe d'un point de vue algorithmique, de sorte qu'un support est nécessaire pour augmenter le niveau d'abstraction lors de la définition de la coordination.

La présentation, **basée sur les travaux de thèse de Giovanni Liboni** tendra à montrer qu'il est possible 1) de fournir une interface de coordination de niveau modèle qui englobe à la fois les modèles cyber et les modèles physiques d'un système cyber-physique et 2) de définir un langage dédié à la coordination de tels modèles. Sur la base de ces artefacts et d'une interface de co-simulation originale, il est possible de générer automatiquement une infrastructure distribuée précise et performante pour la co-simulation.

Cette proposition est soutenue par la mise en œuvre de deux langages dédiés. Un pour la définition de l'interface de coordination du modèle et un pour la définition de la spécification de la

*Intervenant

coordination. Un prototype d'API respectueux de la sémantique comportementale des modèles, qui est introduite comme une généralisation des API standard existantes étaie la proposition. Enfin, un compilateur a été défini pour que les coordinations définies avec les langages proposés puissent être utilisées pour générer automatiquement un environnement de co-simulation distribué. Les différentes propositions sont appliquées sur une étude de cas où les avantages de l'approche sont clairement illustrés.

Session GT Logiciel Éco-Responsable

Green Is the New Clean

Olivier Le Goaer * [1]

[1] LIUPPA – Université de Pau et des Pays de l'Adour : EA3000 – France

Tout comme le *Clean Code*, le *Green Code* est en train de devenir incontournable dans le développement logiciel. Après avoir introduit la problématique générale, cet exposé présente un catalogue d'Energy Code Smells pour Android ainsi que 2 outils développés pour les détecter automatiquement.

*Intervenant

Retour d'expérience sur l'enseignement du Green IT

Adel Noureddine * [1]

[1] LIUPPA – Université de Pau et des Pays de l'Adour : EA3000 – France

Je partage mon expérience d'enseigner un cours de Green IT en master informatique. Ce cours adopte une démarche de pédagogie active en combinant des analyses de documents, des travaux de mesures et de récolte et d'analyse de données, et de l'écoconception de logiciels.

*Intervenant

Table ronde

Adel Noureddine * [1], Olivier Le Goaer *

[2]

[1] LIUPPA – Université de Pau et des Pays de l'Adour : EA3000 – France
[2] LIUPPA – Université de Pau et des Pays de l'Adour : EA3000 – France

Table ronde sur le logiciel éco-responsable :
- Retour sur les 6 défis du logiciel éco-responsable
- Brainstorming sur les défis, et positionnement de la communauté
- Appel à contribution : qui fait quoi ?

*Intervenant

Présentation et activités du GT

Adel Noureddine * [1], Olivier Le Goaer *

[2]

[1] LIUPPA – Université de Pau et des Pays de l'Adour : EA3000 – France
[2] LIUPPA – Université de Pau et des Pays de l'Adour : EA3000 – France

Présentation et activités du GT Logiciel Eco-Responsable

*Intervenant

Session GT IDM

L'Industrie du futur et le jumeau numérique, une opportunité pour l'IDM

Antoine Beugnard * [1], Joël Champeau *

[2]

[1] IMT Atlantique Bretagne-Pays de la Loire (IMT Atlantique) – IMT anlantique : IMTAtlantique – Campus Brest : Technopôle Brest-Iroise CS 8381829238 BREST Cedex 3 -Campus Nantes : 4, rue Alfred Kastler- La chantrerie 44300 NANTES -Campus Rennes : 2 Rue de la Châtaigneraie, 35510 CESSON SEVIGNE, France

[2] École Nationale Supérieure de Techniques Avancées Bretagne (ENSTA Bretagne) – ENSTA Bretagne – 2 rue François Verny, 29806 Brest cedex 9, France

Nous présentons la notion de jumeau numérique, issue du mouvement de l'industrie du futur. Sa mise en oeuvre repose sur l'utilisation de très nombreux modèles et fournit, de ce fait, un terrain d'expérimentation très intéressant pour l'IDM. Dans ce cadre nous illustrons l'utilisation de modèles de processus pour valider des processus industriels à l'aide d'un model-checker et d'un simulateur.

*Intervenant

L'IDM a 20 ans !

Jean-Michel Bruel * [1]

[1] Institut de recherche en informatique de Toulouse (IRIT) – Université Toulouse - Jean Jaurès – 118 Route de Narbonne, F-31062 Toulouse Cedex 9, France

La présentation retracera les grandes étapes de ces 20 dernières années en IDM et dressera quelques pistes pour le futur à la lumière de la trajectoire actuelle.

*Intervenant

Discussion sur le GT IDM

Eric Cariou * [1], Sophie Ebersold *

[2]

[1] Laboratoire Informatique de lÚniversité de Pau et des Pays de lÁdour (LIUPPA) – Université de Pau et des Pays de l'Adour – France
[2] Institut de recherche en informatique de Toulouse (IRIT) – Université Toulouse - Jean Jaurès – 118 Route de Narbonne, F-31062 Toulouse Cedex 9, France

Discussion sur le fonctionnement du GT IDM et de ses activités.

*Intervenant

Co-simulation par des modèles

Julien Deantoni * [1]

[1] Laboratoire dÍnformatique, Signaux, et Systèmes de Sophia Antipolis (I3S) – CNRS : UMR7271, Université Côte d'Azur (UCA), Polytech Nice-Sophia, Inria Sophia Antipolis - Méditerranée, GEMOC Initiative – 2004 rte des Lucioles (Lagrange L-041), BP93, F-06902 Sophia Antipolis Cedex, France

Les systèmes cyber-physiques sont des systèmes d'ingénierie complexe dans lesquels les parties computationnelles communiquent entre elles et avec les parties physiques décrivant l'environnement. Pour apprivoiser la complexité croissante de ces systèmes, ils sont généralement décomposés en différentes parties qui sont modélisées par différents experts, éventuellement issus de différentes organisations. Ces experts utilisent un langage adapté à leur problème, à la fois syntaxiquement et sémantiquement. À ce stade, d'une part on obtient des modèles exécutables des parties computationnelles et des modèles exécutables des parties physiques et, d'autre part, les modèles exécutables ne doivent pas violer les propriétés intellectuelles lorsqu'ils sont partagés et sont donc généralement partagés en tant qu'unités de simulation en boîte noire. Cependant, pour comprendre le comportement émergeant de l'ensemble du système, il est nécessaire de faire une simulation collaborative ; où les unités de simulation de différentes disciplines sont coordonnées pour échanger des données aux moments opportuns.

Les problèmes avec les approches existantes sont multiples. Premièrement, les unités de simulation exposées sous forme de boîtes noires ne permettent pas de prendre en compte les spécificités sémantiques de chaque modèle. Deuxièmement, les co-simulations sont aujourd'hui principalement basées sur une interface de programmation dirigée par le temps, et il a été montré que de telles interfaces introduisaient, lorsqu'elles sont appliquées à des systèmes cyber-physiques, des "retards artificiels" dus à la co-simulation elle-même. De tels retards impliquent une mauvaise précision qui peut invalider les résultats de la co-simulation. De plus, réduire ces délais de co-simulation implique de mauvaises performances globales de co-simulation. Troisièmement, la définition d'un cadre de co-simulation précis et performant peut être complexe d'un point de vue algorithmique, de sorte qu'un support est nécessaire pour augmenter le niveau d'abstraction lors de la définition de la coordination.

La présentation, **basée sur les travaux de thèse de Giovanni Liboni** tendra à montrer qu'il est possible 1) de fournir une interface de coordination de niveau modèle qui englobe à la fois les modèles cyber et les modèles physiques d'un système cyber-physique et 2) de définir un langage dédié à la coordination de tels modèles. Sur la base de ces artefacts et d'une interface de co-simulation originale, il est possible de générer automatiquement une infrastructure distribuée précise et performante pour la co-simulation.

Cette proposition est soutenue par la mise en œuvre de deux langages dédiés. Un pour la définition de l'interface de coordination du modèle et un pour la définition de la spécification de la coordination. Un prototype d'API respectueux de la sémantique comportementale des modèles,

*Intervenant

qui est introduite comme une généralisation des API standard existantes étaie la proposition. Enfin, un compilateur a été défini pour que les coordinations définies avec les langages proposés puissent être utilisées pour générer automatiquement un environnement de co-simulation distribué. Les différentes propositions sont appliquées sur une étude de cas où les avantages de l'approche sont clairement illustrés.

CoSim20: un environnement pour la simulation collaborative et distribuée

Julien Deantoni * [1]

[1] Laboratoire dÍnformatique, Signaux, et Systèmes de Sophia Antipolis (I3S) – CNRS : UMR7271, Université Côte d'Azur (UCA), Polytech Nice-Sophia, Inria Sophia Antipolis - Méditerranée, GEMOC Initiative – 2004 rte des Lucioles (Lagrange L-041), BP93, F-06902 Sophia Antipolis Cedex, France

Les systèmes cyber-physiques sont des systèmes d'ingénierie complexe dans lesquels les parties computationnelles communiquent entre elles et avec les parties physiques décrivant l'environnement. Pour apprivoiser la complexité croissante de ces systèmes, ils sont généralement décomposés en différentes parties qui sont modélisées par différents experts, éventuellement issus de différentes organisations. Ces experts utilisent un langage adapté à leur problème, à la fois syntaxiquement et sémantiquement. À ce stade, d'une part on obtient des modèles exécutables des parties computationnelles et des modèles exécutables des parties physiques et, d'autre part, les modèles exécutables ne doivent pas violer les propriétés intellectuelles lorsqu'ils sont partagés et sont donc généralement partagés en tant qu'unités de simulation en boîte noire. Cependant, pour comprendre le comportement émergeant de l'ensemble du système, il est nécessaire de faire une simulation collaborative ; où les unités de simulation de différentes disciplines sont coordonnées pour échanger des données aux moments opportuns.

Les problèmes avec les approches existantes sont multiples. Premièrement, les unités de simulation exposées sous forme de boîtes noires ne permettent pas de prendre en compte les spécificités sémantiques de chaque modèle. Deuxièmement, les co-simulations sont aujourd'hui principalement basées sur une interface de programmation dirigée par le temps, et il a été montré que de telles interfaces introduisaient, lorsqu'elles sont appliquées à des systèmes cyber-physiques, des "retards artificiels" dus à la co-simulation elle-même. De tels retards impliquent une mauvaise précision qui peut invalider les résultats de la co-simulation. De plus, réduire ces délais de co-simulation implique de mauvaises performances globales de co-simulation. Troisièmement, la définition d'un cadre de co-simulation précis et performant peut être complexe d'un point de vue algorithmique, de sorte qu'un support est nécessaire pour augmenter le niveau d'abstraction lors de la définition de la coordination.

La présentation, **basée sur les travaux de thèse de Giovanni Liboni** tendra à montrer qu'il est possible 1) de fournir une interface de coordination de niveau modèle qui englobe à la fois les modèles cyber et les modèles physiques d'un système cyber-physique et 2) de définir un langage dédié à la coordination de tels modèles. Sur la base de ces artefacts et d'une interface de co-simulation originale, il est possible de générer automatiquement une infrastructure distribuée précise et performante pour la co-simulation.

Cette proposition est soutenue par la mise en œuvre de deux langages dédiés. Un pour la définition de l'interface de coordination du modèle et un pour la définition de la spécification de la

*Intervenant

coordination. Un prototype d'API respectueux de la sémantique comportementale des modèles, qui est introduite comme une généralisation des API standard existantes étaie la proposition. Enfin, un compilateur a été défini pour que les coordinations définies avec les langages proposés puissent être utilisées pour générer automatiquement un environnement de co-simulation distribué. Les différentes propositions sont appliquées sur une étude de cas où les avantages de l'approche sont clairement illustrés.

AFSEC

Une approche polyédrique pour la vérification SMT de réseaux de Petri

Nicolas Amat * [1]

[1] LAAS-CNRS, Université de Toulouse – CNRS – Toulouse, France

Nous définissons une méthode pour tirer parti des réductions de réseaux de Petri en combinaison avec des méthodes de vérification SMT. Nous prouvons la correction de cette méthode en utilisant une nouvelle notion d'équivalence entre les réseaux que nous appelons abstraction polyédrique. Notre approche a été implantée dans un outil, nommé SMPT, qui fournit deux procédures principales : Bounded Model Checking (BMC) et Property Directed Reachability (PDR). Chaque procédure a été adaptée afin d'utiliser des réductions et de travailler avec des réseaux de Petri généraux. Nous avons testé SMPT sur un ensemble d'instances issues de l'édition 2020 du Model Checking Contest. Nos résultats expérimentaux montrent que notre approche fonctionne bien, même lorsque nous n'avons qu'une quantité modérée de réductions.

*Intervenant

SMT-Based Bounded Model Checking of Max-Plus Linear Systems

Muhammad Syifa'ul Mufid * [1], Andrea Micheli [2], Alessandro Abate [1], Alessandro Cimatti [2]

[1] Department of Computer Science, University of Oxford – Oxford, Royaume-Uni
[2] Fondazione Bruno Kessler – Trento, Italie

Max-Plus Linear (MPL) systems are an algebraic formalism with practical applications in transportation networks, manufacturing and biological systems. MPL systems can be naturally modeled as infinite-state transition systems, and exhibit interesting structural properties (e.g. periodicity or steady-state), for which analysis methods have been recently proposed. In this talk, we tackle the open problem of specifying and analyzing user-defined temporal properties for MPL systems. We propose Time-Difference LTL (TDLTL), a logic that encompasses the delays between the discrete-time events governed by an MPL system, and characterizes the problem of model checking TDLTL over MPL. We propose a family of specialized algorithms leveraging the periodic behaviour of an MPL system. We prove soundness and completeness, showing that the transient and cyclicity of the MPL system induce a completeness threshold for the verification problem. The algorithms are cast in the setting of SMT-based verification of infinite-state transition systems over the reals, with variants depending on the computation of the bound, and on the unrolling of the transition relation. Our comprehensive experiments show that the proposed techniques can be applied to MPL systems of large dimensions and on general TDLTL formulae, with remarkable performance gains against a dedicated abstraction-based technique and a translation to the nuXmv symbolic model checker.

*Intervenant

DEPS : A language for modeling and solving system synthesis problem

Pierre-Alain Yvars * [1], Laurent Zimmer *

[2]

[1] Laboratoire QUARTZ EA 7393 – ISAE-Supméca – 3 rue Fernand Hainaut – 93407 Saint Ouen Cedex, France
[2] Dassault Aviation – 78 quai Marcel Dassault – 92552 Saint-Cloud cedex, France

We propose in this talk an approach oriented to the description of the problem to be solved through an adapted formalism called DEPS for doing architecture and system synthesis. DEPS (Design Problem Specification) addresses problems of sizing, configuration, resource allocation/deployment and more generally of architecture generation or synthesis encountered in system design. The systems considered can be physical systems, software-intensive systems or mixed systems (embedded, cyber-physical). This language combines structural modeling features specific to object-oriented languages with problem specification features from constraint programming. We also present an integrated approach through the DEPS Studio environment, allowing DEPS modeling, model compilation and solving using an integrated constraint programming solver on mixed domains.

*Intervenant

Posters et Demos

Programmez vos IHM avec *Interacto*: une démonstration

Arnaud Blouin and Jean-Marc Jézéquel

Univ Rennes, INSA Rennes, Inria, CNRS, IRISA, Rennes
{prénom.nom}@irisa.fr

1 Introduction

Cette démonstration présente *Interacto* [1], une approche pour programmer la partie interactive des interfaces humain-machine (IHM). Cette partie interactive des IHM consiste principalement à traiter les interactions réalisées par les utilisateurs pour les transformer en commandes qui vont modifier ou contrôler le logiciel sous-jacent. Actuellement, les développeurs utilisent principalement une technique proposée avec SmallTalk et le modèle MVC (*Modèle-Vue-Contrôleur*) dans les années 80 : le modèle de traitement des événements IHM. Ce modèle considère les événements IHM de bas niveau (clic souris, *etc.*) comme le concept de première classe que les développeurs peuvent utiliser pour coder et utiliser des interactions utilisateur de plus en plus sophistiquées. Pour les interactions complexes, telles que le drag-lock (un type spécifique de glisser-déposer), le modèle actuel de traitement des événements présente des défauts critiques d'ingénierie logicielle qui entravent la réutilisation et la testabilité du code, et impactent négativement la séparation des préoccupations ainsi que la complexité du code.

Dans [1], nous avons proposé la contribution suivante en matière de génie logiciel : un modèle de traitement des interactions utilisateur appelé *Interacto* qui permet de résoudre les défauts du modèle classique mentionnés ci-dessus.

Cette démonstration a pour but d'expliquer comment *Interacto* fonctionne. Nous utiliserons son implémentation Web (TypeScript/Angular) pour dérouler les différents scénarios utilisés lors de l'étude empirique avec des étudiants dans [1].

2 Présentation d'*Interacto*

Interacto réifie les interactions utilisateur et les commandes IHM en tant qu'objets de première classe et fournit des algorithmes dédiés, des propriétés orientées objet, des optimisations d'exécution et des moyens de test pour permettre aux développeurs de rester concentrés sur les tâches principales de codage des IHM : (i) sélectionner les interactions utilisateur qu'ils doivent utiliser ; (ii) coder comment transformer ces interactions utilisateur en commandes IHM annulables ; (iii) réutiliser les interactions et les commandes ; (iv) écrire des tests IHM. Dans ce modèle, les événements IHM sont désormais considérés comme des concepts d'implémentation de bas niveau rarement utilisés par les développeurs.

Interacto prend la forme d'une API fluente via laquelle un *binder* permet de configurer un *binding Interacto*. Un *binding Interacto* transforme un exécution d'une interaction en une éventuelle commande, comme l'illustre le Listing 1.1.

```
binder()
  .using(() => new DragLock()) // Utilisation d'une interaction drag-lock
  .on(node) // sur l'objet graphique 'node'
  // Pour produire une commande Translate (pour déplacer 'node').
  // La commande Translate est une classe définie par le développeur
  .toProduce(i => new Translate(i.src.clientX, i.src.clientY))
  // Uniquement si le bouton principal de la souris est utilisé
  .when(i => i.button === 1)
  // Pendant l'exécution de l'interaction, mettre à jour la translation
  .then((i, c) => c.setCoord(
     c.shape.x + i.tgt.clientX-i.src.clientX,
     c.shape.y + i.tgt.clientY-i.src.clientY))
  .bind(); // Termine la configuration pour créer un binding Interacto
```

Listing 1.1. Configuration d'un *binding Interacto* pour déplacer un objet graphique

3 Description de la Démonstration

La démonstration impliquera trois scénarios de développement couvrant différentes interactions plus ou moins sophistiquées. Nous nous en servirons pour montrer les problèmes de génie logiciel qui affectent actuellement le modèle de traitement d'évènements IHM.

- Le premier exemple utilise un triple clic sur un objet HTML pour changer sa couleur (stockée dans un service). Nous monterons ensuite comment rendre cette commande annulable (*undo*/*redo*).
- le second exemple se focalise sur une interaction plus sophistiquée, largement utilisée dans les éditeurs de texte mais non fournie sur étagère par les librairies IHM. Cette interaction clavier consiste à attendre un laps de temps d'inactivité clavier (*e.g.* 2 secondes) avant de produire une commande.
- Le troisième exemple consiste à utiliser un glisser-déposer ou un drag-lock pour déplacer un objet HTML dont les coordonnées sont stockées dans un service. Nous monterons ensuite comment rendre cela annulable.

La version TypeScript/Angular d'*Interacto* est disponible librement sur les dépôts NPM[1]. La démonstration utilisera un dépôt Github contenant une application Angular dédiée que tout intervenant pourra télécharger[2].

References

1. Arnaud Blouin and Jean-Marc Jézéquel. Interacto: A Modern User Interaction Processing Model. *IEEE Transactions on Software Engineering*, pages 1–20, 2021. URL: https://hal.inria.fr/hal-03231669.

[1] https://www.npmjs.com/package/interacto
[2] https://github.com/interacto/demo-session

Empirical software engineering

From code similarity detection to model driven similarity detection: first milestones

Jarod BLAIN[1], Corentin GUILLEVIC[1], Alex MOULIN[1], Ali BENJILANY[2]

[1]Université de Nantes, CNRS, LS2N, F-44000, Nantes, France

[2]Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes, Rabat, Maroc

[1] Etudiants en M1 ALMA- [2] Elève ingénieur

Model Driven Software Engineering

Objectives

- Detect similarities in software applications:
  - source code level
  - model level
- Identify the best tool.
- Produce similarity metrics.
- Provide guidelines

Scope

- Object-oriented code similarity is the similarity between syntax, function output, code structure, architecture.
- Comparing code is useful in a lot of situations (see Use Cases) and used in education, engineering, research, etc
- Structure analysis is very effective but difficult due to its specific aspect.
- The state of the art shows that every software has a flaw when it comes to overall comparison (not based on one use case).
- The ultimate goal is to have one tool that can do it all (a do-it-all tool) achieved by building a collaborative toolbox, picking the best from existing soft compare code.
- Going from source code to models.

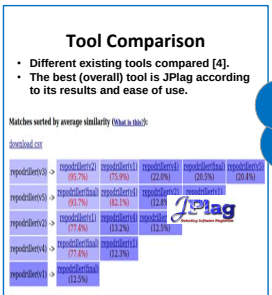
Insights

- Clones and duplicates increase software tests and maintenance costs.
- Factoring out duplicates and inconsistencies management needs duplicates detection that can lean on similarity calculation.
- Experimentations done on the code of the Benchmarks according to Use Cases.
- 15 tools identified and 7 tested.
- State of the art: 23 papers studied and spread out into 5 categories: tool comparison, tools, metrics, methods & approaches, and models.
  - *"Plagiarism detection must be integrated into the toolset and activities of MDE instructors."* [1]
  - *"When source code is copied and modified, which code similarity detection techniques or tools get the most accurate results?"* [3]

Milestone 1 - From code similarity detection...

Tool Comparison

- Different existing tools compared [4].
- The best (overall) tool is JPlag according to its results and ease of use.



"A metric is an information that can be extracted from code resulting in a number (e.g. number of lines)".

Metrics Study

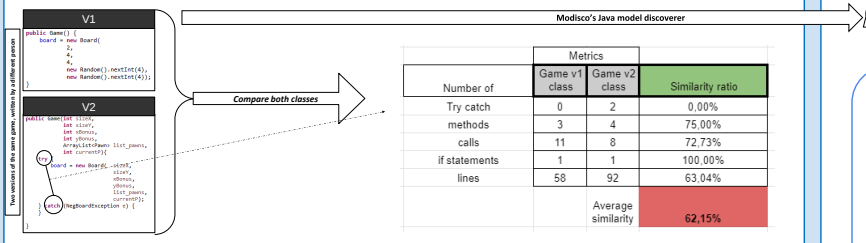
- Similarity ratio = f(software engineering metrics, similarity metrics).
- Metrics help also for abstraction means.
- We experimented a simplified version to validate our computation formula.

%	v1	v2	v3	v4	v5	final
v1	73.93	80.64	70.71	24.31	28.10	
v2		89.28	24.32	21.22	19.06	
v3			15.48	19.21	22.57	
v4				80.85	74.47	
v5					92.92	

%	v1	v2	v3	v4	v5	final
v1		86.14	87.34	54.64	51.08	48.10
v2			78.57	43.78	41.79	40.12
v3				60.80	55.91	52.02
v4					87.03	78.75
v5						90.10

%	v1	v2	v3	v4	v5	final
v1		83.55	94.19	42.38	39.13	34.92
v2			79.64	28.52	26.35	23.53
v3				47.80	44.13	39.37
v4					89.24	80.40
v5						89.85

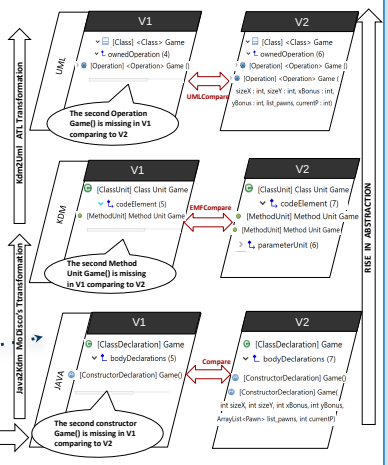
A SimpleGame Example (code similarity)



	Metrics		
Number of	Game v1 class	Game v2 class	Similarity ratio
Try catch	0	2	0,00%
methods	3	4	75,00%
calls	11	8	72,73%
if statements	1	1	100,00%
lines	58	92	63,04%
		Average similarity	62,15%

Plagiarism detection requires code or model mutation to introduce the malicious copy strategy (future work).

Milestone 2 - ... to model similarity detection



- Same results, different entities : (Java → Constructor - KDM model → MethodUnit - Operation<--> UML).
- Model similarity detection reveals changes made on the architecture of a program regardless of code details thanks to the rise in abstraction, while the code similarity detection returns a global rate of similarity.
- *Abstraction requires fine heuristics to obtain high-level concepts* [2].

→ Meanwhile, the upcoming milestones of the research might come up with a proof of the efficiency of a certain model-based comparison compared to the others. Much remains to do, especially in reverse engineering and comparison implementation.

References

[1] Salvador Pérez, Manuel Wimmer, and Jordi Cabot. "Efficient plagiarism detection for software modeling assignments". In:Computer Science Education30 (Jan. 2020), pp. 1–29.doi:10.1080/08993408.2020.1711495

[2] Pascal André et al. "JavaCompExt: Extracting Architectural Elements from Java Source Code".In:WCRE. Lille, France: IEEE, Oct. 2009, pp. 317–318.doi:10.1109/WCRE.2009.53.

[3] Chaiyong Ragkhitwetsagul, Jens Krinke, and David Clark. "A comparison of code similarity analysers". In:Empirical Software Engineering23.4 (2018), pp. 2464–2519.

[4] Master TER Report 2021 https://uncloud.univ-nantes.fr/index.php/s/k2Ezp4bJ8bki3Sn

Use Cases

- Plagiarism
- Licensing
- Clone detection
- Code refactoring & Code reuse
- Pattern detection
- Software defects,
- Malware insertion

Best tools: JPLAG, MOSS

According to our experimentations, JPLAG and MOSS are very good in their comparison results, they both give accurate detailed and easy to understand results. JPlag is used in this poster for it's easiness and simplicity.

Benchmarks

- Simple programs
- Student applications
- Lego EV3+android applications
- Git repositories
- Plagiarism benchmarks

Further information

Supervisors: ANDRE Pascal, BRUNELIERE Hugo, TAMZALIT Dalila

Students report: https://uncloud.univ-nantes.fr/index.php/s/k2Ezp4bJ8bki3Sn

https://www.ls2n.fr/equipe/aelos/

https://www.ls2n.fr/equipe/naomod/

Poster credit - https://colinpurrington.com/tips/poster-design/

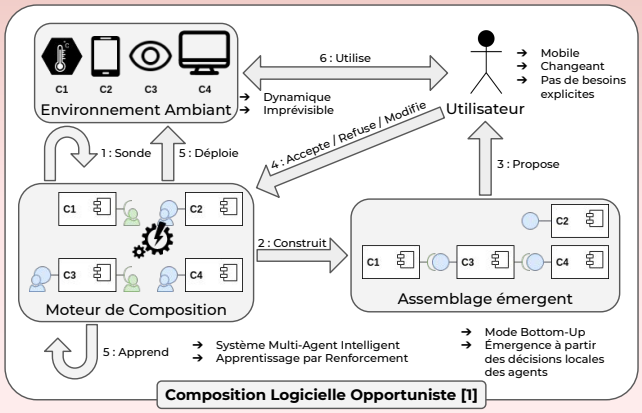
Composition Opportuniste en Milieu Ambiant supportée par Ligne de Produits Logiciels

Kévin DELCOURT, Françoise ADREIT, Jean-Paul ARCANGELI, Sylvie TROUILHET

prenom.nom@irit.fr

Institut de Recherche en Informatique de Toulouse - Université de Toulouse - UT2J - UPS

Contexte

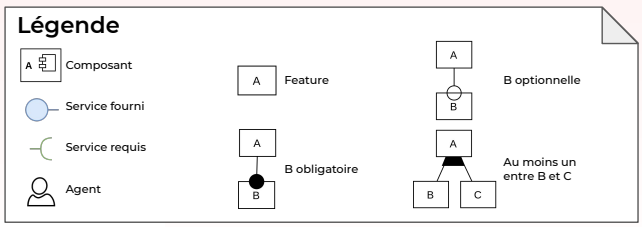


Problématique

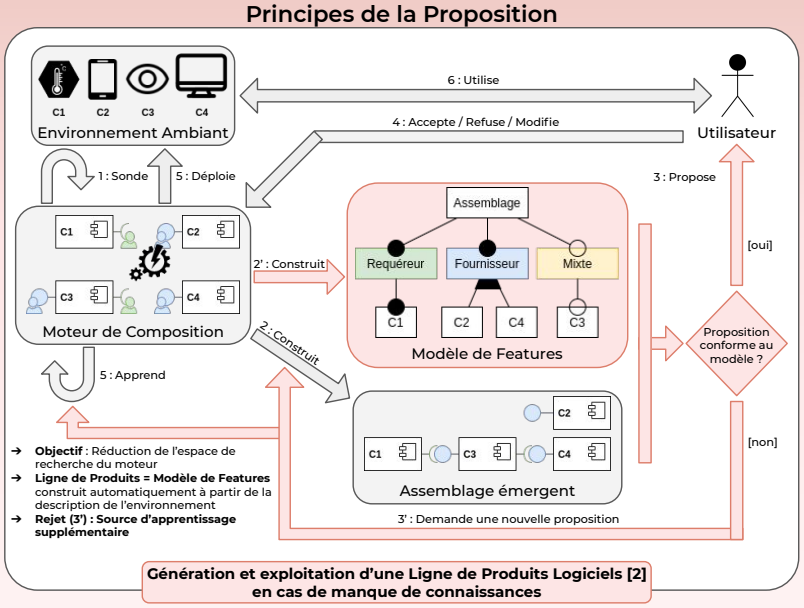
Cas de situations jamais encore rencontrées : Le moteur ne dispose pas de connaissances et fait des choix aléatoires de composition.

➔ **Comment proposer un assemblage pertinent pour l'utilisateur ?**

Légende



Principes de la Proposition



Travaux en cours

➔ **Consolidation** de la solution de génération et d'exploitation du modèle de features

➔ **Exploration** de différentes méthodes de construction du modèle de features

➔ **Étude comparative** et **validation** de la solution par rapport à des cas d'utilisation

Références

[1] Younes, W. (2021). *Un moteur intelligent pour la composition logicielle opportuniste dans les environnements ambiants et mobiles.* PhD thesis, Université de Toulouse, UPS.

[2] Metzger, Andreas, et al. (2019). *Feature-Model-Guided Online Learning for Self-Adaptive Systems.* arXiv preprint arXiv:1907.09158.

Reflectivipy: building python debuggers with sub-method, partial behavioral reflection

Steven Costiou[1], Vincent Aranega[2], and Marcus Denker[1]

[1] Inria, Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
[2] Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
{steven.costiou,vincent.aranega, marcus.denker}@inria.fr

Abstract. Building debugging tools is hard and requires powerful tools and libraries. In object-oriented technologies, it is common to use fine-grained reflection to implement debuggers. In this tool presentation, we describe how partial behavioral reflection applied to sub-elements of a method helps in the implementation of advanced debugger features. As an example, we present an implementation of object-centric breakpoints in python.

Keywords: Debugging · Object-centric · Python · Reflection

1 Introduction

Debugging is a general concern in software engineering. Therefore we need to build debuggers, and for that we require support from languages and their infrastructure. Sub-method, partial behavioral reflection [1] is a reflection technique for fine-grained instrumentation of object-oriented programs.

The technique consists in annotating AST nodes at run time with *metalinks*. Metalinks describe calls to a meta-object to be executed for the operation defined by the AST node. They define when to call (before, after, instead) and which information to reify and pass to the meta-object. That meta-object implements and executes instrumentation behavior, such as debugging operation.

Installing metalinks leads to the code being dynamically transformed, recompiled, and installed. This allows the system to be annotated at run time (Figure 1).

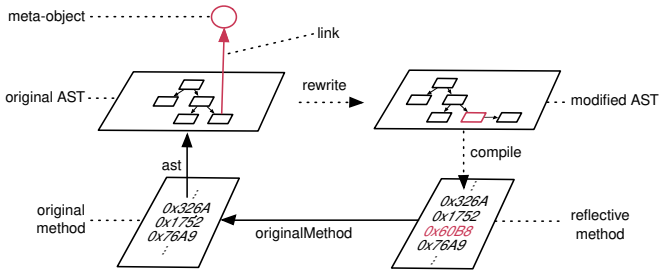


Fig. 1: Run-time method instrumentation with sub-method, partial behavioral reflection

The technique has successfully been implemented and integrated with Pharo as Reflectivity [1]. It has been used in more than 20 research projects to implement fine-grained and customized program instrumentation. Many of these projects are about designing and implementing new debugging tools. In all these projects, sub-method, partial behavioral reflection is the fundamental support for implementing debuggers. In this paper, we present Reflectivipy, our python implementation of the technique. We briefly describe its API, and how it is used to implement an advanced debugger able to target one specific object in a running program. The debugger provides breakpoints scoped to a specific object, with two granularities: on a method or on any sub-expression of a method. The debugger relies on Reflectivipy and IPDB[3].

2 Reflectivipy in a Nutshell

Reflectivipy is our python implementation of sub-method, partial behavioral reflection [1][4]. The flow for the creation and installation of a metalink using `reflectivipy` is the following:

[3] https://ipython.readthedocs.io/en/stable/api/generated/IPython.core.debugger.html
[4] https://github.com/StevenCostiou/reflectivipy

(1) Select an AST node of a method using: reflective_ast_for_method(*classOrObject*, *methodName*).
(2) Create a metalink and configure it using:
MetaLink(*meta-object*, *meta-objectMethodName*, *"before"|"instead"|"after"*).
(3) Install the metalink on the selected AST node using: link(*configuredLink*, *aSelectedNode*).

3 Object-centric Debugging for Python

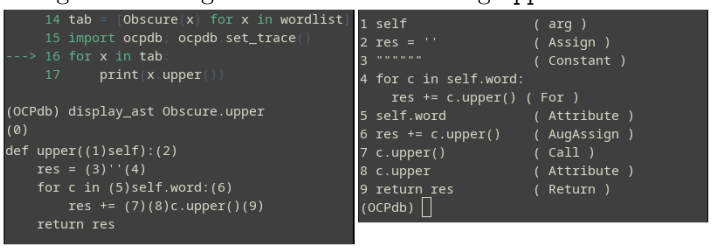
Object-centric debugging [2] scopes breakpoints to specific objects and their interactions instead of breaking the execution for all objects of the same kind. This helps developers to focus their debugging investigations to precise parts of their program. In this demonstration, we illustrate how we use Reflectivipy to implement a breakpoint that interrupts an execution when a precise object receives a message. This breakpoint builds a meta-object that breaks the execution, configures a metalink to call that meta-object, and installs this metalink on an AST of an object's method.

Implementing an object-centric breakpoint with Reflectivipy. To build our object-centric breakpoint, we attach a metalink to the object we want to debug. We install the metalink on a (sub)expression (*i.e.*, an AST node) of a method bound to the object to debug. When the metalink is activated, *e.g.*, when the execution reaches the aforementioned expression, it automatically calls the Python special method `set_trace()` that interrupts the execution.

Installing an object-centric breakpoint. We first have to set a first breakpoint to interrupt the execution and activate the object-centric debug mode. To do that, we insert the breakpoint instruction `ocpdb.set_trace()` in the code we want to debug.

The command `display_ast` command (Figure 2) labels all nodes of a method with a number. Each number correspond to a given AST node. We can install breakpoints on a statement (*e.g*, AST node 6) or on an expression (*e.g*, AST node 7).

Fig. 2: Labelling the AST for an existing `upper` method.



With the `halt` command, we install an object-centric breakpoint. It takes two arguments: (1) the method of a specific object and (2) the number of the node where installing the breakpoint in that method. Let us use it to debug a program that transforms a list of names from lowercase to uppercase. The program output produces ALBERT JODIE MIKE JOHN CARMEN NaLLELY GINA. One of the name is not transformed correctly. To debug it, we install an object-centric breakpoint on the nallely string object to interrupt the execution when upper is called on that object: halt tab[5].upper, 6

We resume the execution which stops exactly on the sixth element of the list (the nallely string object). A program state inspection reveals that the second lowercase letter is not a latin letter, but an UTF-8 letter looking like a latin letter. On a large collection and in a complex program, stopping on the right object is tedious without object-centric breakpoints. In that case, we have to express breakpoint conditions to find the object, which is difficult.

4 Conclusion

We presented Reflectivipy, our Python implementation of sub-method partial behavioral reflection. We used it to implement a breakpoint that scopes to specific objects with a sub-expression granularity. The technique offers strong support for build debugging tools. However, exposing ASTs is tedious and we do it by labeling nodes which is impractical. We need more tools dedicated to ASTs to support the building debuggers based of sub-method, partial behavioral reflection.

References

1. S. Costiou, V. Aranega, and M. Denker. Sub-method, partial behavioral reflection with reflectivity: Looking back on 10 years of use. *The Art, Science, and Engineering of Programming*, 4(3), Feb. 2020.
2. J. Ressia, A. Bergel, and O. Nierstrasz. Object-centric debugging. In *Proceeding of the 34rd international conference on Software engineering*, ICSE '12, 2012.

Supporting Library Evolution with Smart Deprecations that Automatically Fix Client Code

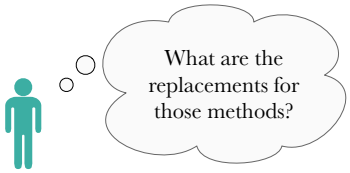
Before

1 Library developers deprecate certain methods. Sometimes they delete methods without deprecation.

```
@Deprecated
void method(…)
```

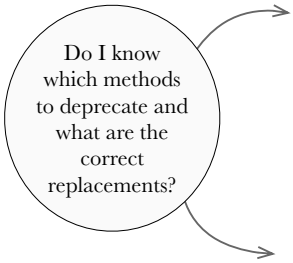
2 To update their software to the new version of a library, client developers must identify the correct replacements for deleted and deprecated methods.

500 warnings



Now

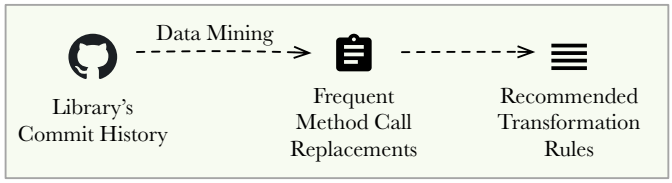
1 Library developers can write transformation rules to specify replacements for deprecated methods.



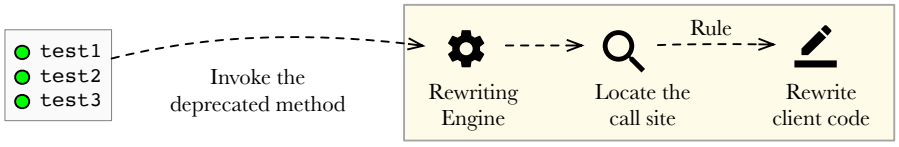
Write deprecations and transformation rules manually

```
@Deprecated
@Transform(method -> newMethod)
void method(…)
```

Ask DepMiner tool to propose deprecations and generate the rules



2 Client developers only have to execute their code or run the tests. When a deprecated method is called, the DepRewriter tool automatically identifies the call site and fixes client code using the transformation rule. Execution is not interrupted.



Oleksandr Zaitsev[1,2], Stéphane Ducasse[2], Nicolas Anquetil[2], Arnaud Thiefaine[1]
[1]Arolla, Paris, France
[2]Inria, Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL, Lille, France

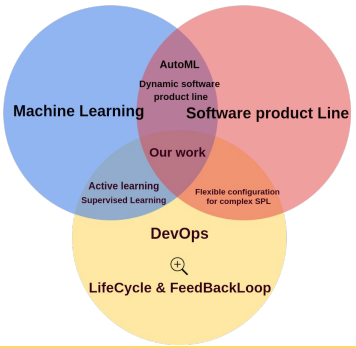
UNIVERSITÉ CÔTE D'AZUR

EZAKO Intelligence in data

Maximization of expert feedback in the detection of anomalies in time series: from the formalization of workflows to their operationalization.

Yassine EL Amraoui, Mireille Blay-Fornarino, Frédéric Precioso, EZAKO

Given that a domain expert comes with her data organized in time series and aims to detect anomalies within it, our objective is to automatically suggest one suitable machine learning workflow with the minimal user interactions



The colors indicate each boxes related research field

Data Scientist

Knowledge and practices of machine learning anomaly detection, evolving, contradictory, and partial.

Capitalize on knowledge in the form of Business Processes, Feature models and numeric constraints in a non-closed world [1][2]

2

Data Scientist

How to identify the anomaly detection problem by making the best use of the expert's knowledge and the data ?

The questions are linked to the features. Only the minimal set of critical questions are presented to the domain expert

Domain expert

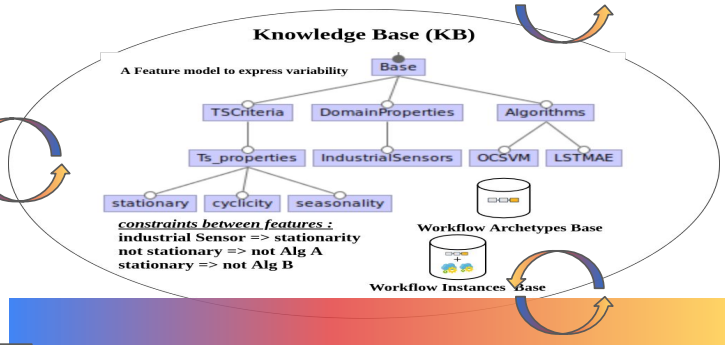
3

The Domain expert presents the datasets and domain constraints to the best of his knowledge

Domain expert

Dataset + Domain knowledge + Domain contraints

1



KB master

Frequently analyse the knowledge base to **detect inconsistencies and make them explicit,** by taking into account the addition of new solutions,and verifying that information is not lost.[3]

Frequently feed the knowledge base with information found in the litterature

Literature

Data Scientist

Run multiple experimentations in attempt to solve the anomaly detection problem

- **Integrate custom workflows**
- **Identify variability points**
- **Express new constraints**

6

The system suggests :

- **A unique machine learning workflow**
- **Interesting configurations and prohibited ones**

Config 1 Config 2 Config 3

4

Domain expert

- **Label data if necessary**
- **Validate the results**
- **Make sure all domain constraints are respected**

5

References:

[1] Cognini R., Corradini F., Polini A., Re B. (2015) Extending Feature Models to Express Variability in Business Process Models. In: Persson A., Stirna J. (eds) Advanced Information Systems Engineering Workshops. CAiSE 2015. Lecture Notes in Business Information Processing, vol 215. Springer, Cham. https://doi.org/10.1007/978-3-319-19243-7_24

[2] Grimm S., Motik B., Preist C. (2006) Matching Semantic Service Descriptions with Local Closed-World Reasoning. In: Sure Y., Domingue J. (eds) The Semantic Web: Research and Applications. ESWC 2006. Lecture Notes in Computer Science, vol 4011. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11762256_42

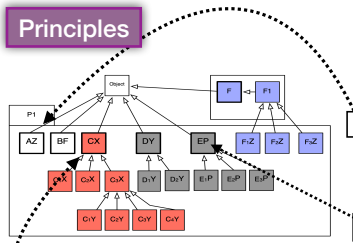
[3]David Benavides, Sergio Segura, Antonio Ruiz-Cortés,Automated analysis of feature models 20 years later: A literature review,Information Systems,Volume 35, Issue 6, 2010,Pages 615-636,ISSN 0306-4379,https://doi.org/10.1016/j.is.2010.01.001.

[4]Fabijan A., Olsson H.H., Bosch J. (2015) Customer Feedback and Data Collection Techniques in Software R&D: A Literature Review. In: Fernandes J., Machado R., Wnuk K. (eds) Software Business. ICSOB 2015. Lecture Notes in Business Information Processing, vol 210. Springer, Cham. https://doi.org/10.1007/978-3-319-19593-3_12

ClassNames Distribution: Detecting inconsistencies in class names

Nour Jihene Agouf, Stéphane Ducasse, Anne Etien, Alidra Abdelgani
Working groups: CLAP, GLIA

Principles

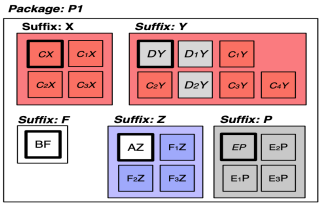


A schematic mini project which shows hierarchies types and their colors

Class or trait ➔ Belongs to no hierarchy

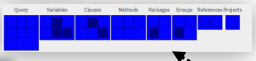
One hierarchy ➔ One suffix

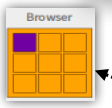
One hierarchy ➔ More than one suffix- a color means a hierarchy



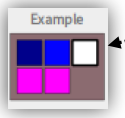
The ClassNames Distribution package for the previous schematic mini project

Concrete case





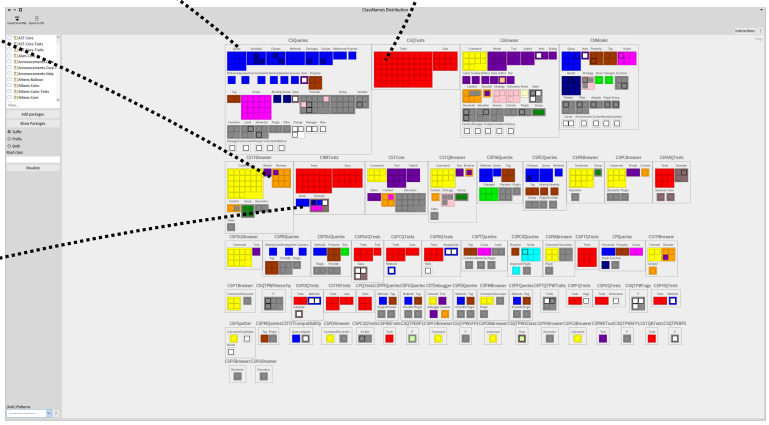
Intruder: a hierarchy which uses the same suffix as an other hierarchy in the same package



Confetti: Multiple hierarchies use the same suffix in the same package

Scattered-Vocabulary: One hierarchy uses several suffixes in the same package

Blob: a hierarchy which uses more than one suffix but makes the most use of the 'Tests' suffix in this case



ClassNames Distribution tool user interface on the Calypso v6 project

About the tool

- Supports both Pharo and Java projects.
- Supports suffix analysis, prefix analysis and both of suffix and prefix mixed to chose the most accurate.
- Implemented in Pharo and is available here: https://github.com/NourDjihan/ClassNameAnalyser