Rapport de conjoncture

GDR Génie de la Programmation et du Logiciel

Coordonné par Philippe Collet (I3S), Lydie Du Bousquet (LIG), Laurence Duchien (LIFL)
à partir des contributions des groupes du GDR GPL et de réponses à l'appel à défis 2025

Introduction

Le GDR Génie de la Programmation et du Logiciel (GPL) a lancé en janvier 2014 un appel à défis 2025, qui fait suite à l'appel à défis 2020 lancé en 2010 et dont un texte a été publié dans TSI en 2012[1]. Nous reprenons ici une partie des réponses à ce dernier appel à défis, tout en le complétant par d'autres travaux de façon à aborder l'ensemble des thèmes de la communauté GPL.

Les chercheurs du GDR travaillent sur des abstractions logicielles et les fondements scientifiques associés, en vue de proposer des solutions bien fondées et pratiques permettant la production et la maintenance de logiciels. Ces abstractions logicielles interviennent à toutes les étapes du cycle de vie du logiciel, de sa conception à son exécution, et prennent forme dans des théories, techniques et méthodes de modélisation, de validation, de vérification, dans les langages de programmation, dans les plates-formes d'exécution ou encore lors de la mise en place de tests. Quelle que soit l'étape du cycle de vie, l'objectif est d'offrir des moyens de construction de logiciel fiable, évolutif, tout en apportant une aide aux concepteurs et en maîtrisant coûts et délais.

Les problématiques sur lesquelles les chercheurs se positionnent sont renouvelées par de nouveaux domaines d'application dans tous les secteurs (informatique embarquée, intelligence ambiante, informatique dans les nuages), de nouveaux enjeux de société (développement durable, économies d'énergie, objets connectés), ainsi que par la diversité des fonctionnalités à fournir et à adapter aux besoins spécifiques de clients et aux modifications continuelles.

Les questions auxquelles les chercheurs du GDR tentent de répondre sont **:**

- **Comment concevoir et réaliser des systèmes personnalisés ou capables de s'adapter par eux-mêmes, en pouvant facilement découvrir, sélectionner et intégrer les services disponibles à l'exécution ?**
- **Comment valider, vérifier et tester ces logiciels qui vont évoluer dans des environnements qui deviennent de plus en plus imprédictibles ?**

Des exigences à la réalisation des logiciels

Concevoir un logiciel en fonction des besoins d'un client et l'adapter aux besoins d'un autre, en réutilisant une partie du code, ne peut plus se faire de façon artisanale. L'approche proposée par les lignes de produits logiciels permet de généraliser, et donc de maîtriser, la variabilité des logiciels. Cependant, la gestion et la modélisation de la variabilité à grande échelle restent un défi. Dès lors qu'il faut gérer des milliers, voire des millions de configurations de produits logiciels, l'identification, la définition et la composition des variants se doivent d'être accompagnées de méthodes et d'outils permettant le passage à l'échelle.

Un second enjeu réside dans la diversification et la spécialisation des langages utilisés par les acteurs impliqués dans la construction des logiciels. Ces langages sont spécifiques à un domaine (DSL ou *Domain-Specific Language*), permettant ainsi à un expert de développer rapidement, et de façon fiable, un logiciel adapté à son expertise. Actuellement, le développement de logiciels demande de nombreuses expertises. L'un des défis est de composer et coordonner ces langages de modélisation spécifiques de façon à construire des logiciels multi-domaines.

[1] [1] http://tsi.revuesonline.com/article.jsp?articleId=17328

Pour l'ingénierie dirigée par les modèles (IDM), les approches se limitent souvent à une seule dimension de modélisation. Pour prendre en compte la diversité des modèles, il est nécessaire d'augmenter les efforts sur les aspects notation et interaction pour que cette approche soit plus facilement utilisable et apporte une véritable évolution dans les pratiques du développement logiciel. Pour cela, l'un des défis principaux réside dans la capacité à représenter graphiquement des modèles de très grande taille. L'une des pistes envisagées est d'initier un débat interdisciplinaire entre l'IDM, les sciences cognitives, le *design* informatique et l'interaction homme-machine, pour refonder les paradigmes d'écriture des modèles.

La multiplication des systèmes de systèmes et systèmes ubiquitaires nécessite d'adapter ou de repenser les méthodes de conception. Un système de systèmes est une composition de systèmes complexes existants. Un système ubiquitaire doit s'adapter à des configurations d'environnement « mouvantes », où les dispositifs apparaissent et disparaissent. Dans ces conditions, l'un des défis les plus importants est de savoir comment modéliser et analyser les propriétés non fonctionnelles, telles que la sécurité, la fiabilité et la sûreté.

De la compilation à l'exécution des logiciels

Tous les systèmes informatiques, des smartphones jusqu'aux accélérateurs de calcul (GPUs, FPGAs, super-calculateurs des futurs centres de calcul, etc.) sont désormais pourvus de systèmes multi-cœurs. Ces nouveaux matériels mettent le parallélisme à la portée de tous, mais restent d'une grande complexité. Le défi est double. Il faut d'une part maîtriser la difficulté de programmation de ces architectures tout en assurant une certaine portabilité des codes et des performances. Il faut, d'autre part, offrir une meilleure interaction entre utilisateurs et compilateurs. Cela nécessite de revisiter les langages, notamment parallèles, les techniques de compilation (analyse et optimisation de codes), et les systèmes d'exploitation.

L'internet des objets est entravé par le fait que le développement de pilotes de périphériques reste une tâche complexe et source d'erreurs qui exigent un haut niveau d'expertise, à la fois liée à la multiplicité des systèmes d'exploitation cibles et au dispositif visé. Il est nécessaire de proposer de nouvelles méthodologies qui amènent une rupture forte avec les méthodes actuelles de développement de pilotes de périphériques. L'une des pistes envisagées est de s'inspirer du génome dans le domaine de la biologie en étudiant le code du pilote de périphérique existant et de le faire muter vers une version qui s'adapterait à l'environnement cible.

L'omniprésence des logiciels dans notre vie quotidienne engendre une part importante de la consommation énergétique globale française (13%), et la communauté du génie logiciel doit jouer un rôle important. En particulier, les logiciels s'exécutant sur des architectures multi-cœurs exploitent mal leur consommation énergétique. En identifiant les leviers qui peuvent être exploités dans les différentes couches d'un système informatique, il sera possible d'optimiser cette consommation en continu. L'éco-conception des logiciels apparaît donc comme un défi afin de sensibiliser et guider les développeurs dans la réalisation de logiciels efficients.

Méthodes et outils de validation et de vérification

Le test est l'une des méthodes de validation les plus utilisées dans l'industrie. Une meilleure intégration des techniques de tests à partir de modèles et de code - *code-based testing* (CBT) et *model-based testing* (MBT) - a été possible par les langages d'annotation permettant l'ajout d'éléments de modèles dans le code et le contrôle de leur évolution conjointe. Néanmoins, l'expressivité de ces langages reste confinée aux tests unitaires et n'aborde pas la problématique du test de recette. L'utilisation de tests dans les méthodologies, comme les méthodes agiles, pose également des questions telles que l'accroissement du nombre de tests et la pertinence des tests de régression. Il est nécessaire de proposer des méthodes gérant à la fois l'évolution du logiciel et des tests associés. Le passage à l'échelle des techniques de génération de tests est un enjeu majeur.

Du point de vue de la sûreté et la sécurité, l'apparition de *Software as a Service* (SaaS) a amené de nouveaux questionnements. Les techniques de test actif, comme le test de pénétration, assurent en

amont la sécurité du système. Cependant un défi lié à leur développement requiert la connaissance de la logique applicative pour permettre l'exploration exhaustive des points de vulnérabilité potentiels.

En ce qui concerne la vérification de modèles (*model-checking*), l'impact dans l'industrie est principalement limité aux systèmes embarqués critiques. Deux raisons principales sont la réponse binaire à des propriétés de satisfaction, insuffisamment informative, et l'abstraction insuffisante pour répondre au réglage et à l'évolutivité des systèmes. Un défi majeur consiste à surmonter ces limitations en offrant des méthodes formelles paramétriques pour la vérification et l'analyse automatisée du comportement des systèmes, dès la phase de conception.

En ce qui concerne la vérification par preuves, un défi récurrent est aussi de faire pénétrer plus amplement les approches et outils dans le monde industriel. Ceci passe par une meilleure automatisation des preuves, mais ce n'est pas le seul levier. Réutiliser spécifications et preuves associées serait un atout important. Modularité, héritage, paramétrisation facilitent la réutilisation, mais ne suffisent pas. Des approches inspirées des lignes de produit ou des approches *à la carte* ont été proposées, par exemple dans le cadre de Coq, mais ces mécanismes demandent à être simplifiés. Un autre mode de réutilisation serait possible en développant un standard assurant l'interopérabilité entre les différents outils et formalismes, permettant ainsi la réutilisation de preuves *in the small* (dans un formalisme donné) ainsi que la réutilisation des preuves *in the large* (dans différents formalismes).

Finalement, un dernier défi majeur est la validation des systèmes dans des univers flexibles et ouverts. Ces caractéristiques signifient l'imprévisible (nouvelles fonctionnalités, nouveaux usages, évolutions environnementales). Les techniques courantes de validation s'appuient sur des spécifications qui n'évoluent pas. Aussi, il est essentiel de revisiter les solutions de validation pour tenir compte de la diversité et de l'imprédictible.