# Présentation des résultats du Projet ANR AAPG 2018 – PHILAE From Model-Based Testing to Cognitive Test Automation

Bruno LEGEARD (FEMTO-ST), Roland GROZ (LIG), Abbas AHMAD (FEMTO-ST)

May 2022

## 1 Introduction

Le projet ANR Philae - From Model-Based Testing to Cognitive Test Automation, sélectionné dans le cadre de l'appel à projet générique ANR 2018, s'est déroulé d'octobre 2018 à septembre 2022. L'équipe projet est composé de 4 partenaires français : Institut Femto-st/DISC (Coordinateur), Laboratoire d'Informatique de Grenoble, Orange Labs et Smartesting, auxquels se sont joints Mark Utting (actuellement à University of Queensland – Australie) et Arnaud Gotlieb (Simula Research Laboratory – Norvège).

Le problème auquel le projet s'intéresse est celui du coût grandissant des tests de non-régression dans les systèmes logiciels, qui constitue un goulot d'étranglement pour l'intégration continue. L'intuition de base est que la grande quantité de tests et la présence de masses importantes de traces d'exécution enregistrées dans des journaux (logs) sur les plates-formes doit permettre d'apprendre des modèles des systèmes pour réduire et améliorer les efforts de test. La thématique générale du projet est l'analyse par apprentissage automatique des traces d'exécution obtenues à partir des logs pour (1) évaluer et optimiser la couverture réalisée par les tests de l'usage réel de l'application par ses utilisateurs et (2) détecter dans les traces d'exécution de tests des anomalies sur ces tests. Contrairement à une approche de test à base de modèles (connu sous l'appellation Model-Based Testing), dans laquelle les modèles sont préexistants ou spécifiquement créés pour générer les cas de test, l'analyse dans Philae se base sur l'apprentissage machine de modèles (statistiques ou déterministes) à partir des logs.

Les résultats du projet Philae sont concrétisés et disponibles au travers d'une boîte à outils open-source, qui intègre des composants logiciels pour acquérir et formater les traces d'exécution, pour réaliser des analyses sur ces traces, et pour permettre de les utiliser à des fins de génération de tests ou d'analyse de tests. Les éléments open-source mis à disposition de la communauté intègrent aussi deux cas exemples avec les données de traces pour la reproduction des résultats, mais aussi disponibles comme jeux de données ouvertes.

Dans la suite de cet article, en section 2, nous commençons par présenter l'objet de notre recherche : la trace d'exécution. En sections 3 et 4, nous présentons les travaux et résultats obtenus respectivement sur les sujets (1) et (2). En section 5, nous présentons le répertoire de code et de données en open-source Philae mis à disposition de la communauté (sous Github).

## 2 Notion de trace d'exécution

La donnée brute sur laquelle se fonde le projet Philae est le recueil de logs, c'est-à-dire une suite d'événements enregistrés lors de l'exécution du logiciel. Ces logs proviennent des exécutions en production mais aussi des exécutions de tests. Les logs constituent une matière brute dont le premier traitement réalisé consiste à transformer les logs en traces.

Dans Philae, nous appelons *traces d'exécution* une séquence d'événements enregistrés présentant une session cohérente d'exécution. Dans le cas d'une exécution en production, cette session cohérente correspond généralement à une session utilisateur, et dans le cas d'une exécution de tests, cette session correspond à l'exécution d'un test. Le passage des logs aux traces est généralement réalisé par regroupement des événements à partir d'une clé de session portée par les événements enregistrés, mais parfois cela peut être plus complexe (par exemple lors d'un chaînage particulier des événements constituant une trace). Les événements de logs présentent un horodatage : les traces sont donc des séries temporelles, avec chaque événement portant un ensemble d'information.

Un domaine particulièrement ciblé par le projet Philae est celui des applications web, pour lesquels on dispose fréquemment des logs situés aux niveaux des APIs / appels de services, qu'on traitera donc préférentiellement plutôt qu'aux logs sur l'interface humain/machine de l'application.

Pour illustrer le passage des logs aux tests, nous présentons, en Figure 1, un des deux cas exemples en données ouvertes Philae : les logs issus d'une instance de boutique en ligne avec l'API Spree[1].

Nous observons sur la figure 1 une trace utilisateur. Cette trace correspond à des appels API sur la plateforme SPREE lors d'un parcours du client. La trace montre les événements qui font parties d'un scénario lorsque le client arrive sur la boutique en ligne, clique sur un produit "denim-jacket", l'ajoute à son panier et complète le processus de paiement.

Il faut noter qu'une action sur l'interface graphique se traduit généralement par plusieurs appels API. Par exemple, le fait d'ajouter un item au panier sur l'interface utilisateur se traduit par des appels API de récupération/création du panier à l'aide du token d'authentification de l'utilisateur et de l'ajout de cet item au panier récupéré/créé. De plus, les logs représentant la trace contiennent tous les paramètres nécessaires afin d'être rejoué ou utilisé dans un algorithme d'apprentissage machine pour générer de nouvelles traces exécutables.

[1] Cf. `https://spreecommerce.org/`


Figure 1: Extrait des logs Spree commerce API

# 3 Evaluer et optimiser la couverture de l'usage par les tests

Le projet Philae a étudié, développé et expérimenté cinq types de traitements des logs, constituant des composants d'une boîte à outils pour générer des tests fonctionnels automatisés à partir de l'analyse de l'usage :

- Conversion des logs en traces et visualisation des traces sous la forme de workflow et capacité d'exploration des traces[2];
- Codage des traces sous forme de vecteurs pour l'utilisation des données par les algorithmes ML[1] (en utilisant des techniques tel que word2vec issu du TALN – Traitement Automatique du Langage Naturel) ;
- Clustering des traces pour regrouper les traces par patterns d'usage et évaluation des clusters pour mesurer la représentativité des patterns vis-à-vis des comportements du logiciel[3] (par couverture de code en particulier) ;
- Mesure et visualisation de la couverture des traces d'usage par les traces de tests (cf. figure 2), et identification des clusters à couvrir ou mieux couvrir pour augmenter la couverture de l'usage par les tests ;
- Génération de tests représentatifs sur les clusters avec des techniques d'apprentissage machine génératives et génération de scripts exécutables dans un framework open-source d'automatisation des tests sur API[4].

Les techniques d'apprentissage machine utilisées sont, d'une part des techniques dites classiques (telles que les algorithmes de clustering et les arbres de décision), et d'autre part des modèles de Deep Learning, en particulier issus du TALN, tels que les Transformers.

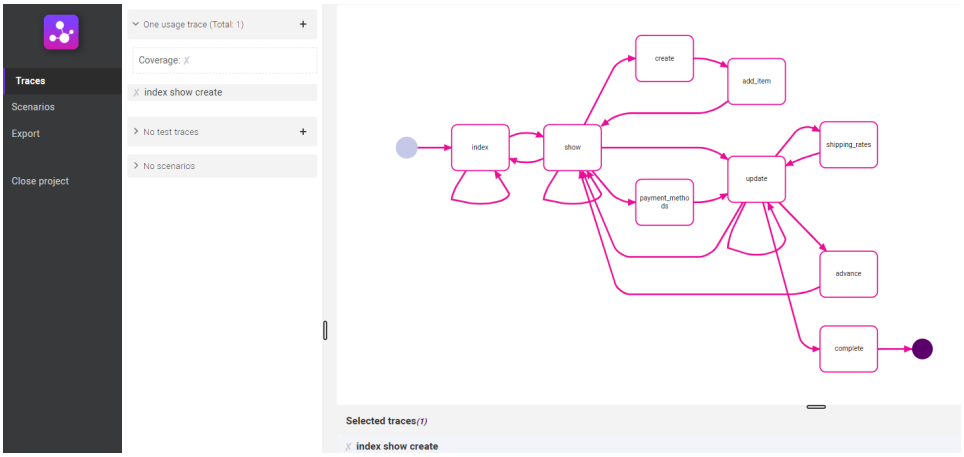


Figure 2: Visualisation des traces d'usage sous la forme de workflow

La figure 2 illustre la visualisation graphique sous la forme d'un workflow représentant un ensemble de traces capturées à partir des logs. C'est une capture d'écran de l'outil Philae de visualisation des traces d'usage.

# 4 Détecter dans les traces d'exécution de tests des anomalies sur ces tests

Le projet Philae s'est intéressé à l'analyse des logs obtenus lors de phases de tests intensives, comme les tests de non-régression ou les tests d'endurance, lesquels visent à vérifier le bon comportement du système pendant des temps longs, pour y détecter d'éventuelles fautes liées à des fuites mémoires ou d'autres accumulations de consommations de ressources. L'exploitation de ces volumes très importants de logs, en l'absence de modèles formels du comportement pouvant servir d'oracles est difficile, car il faut y repérer les comportements déviants et essayer d'en retrouver les causes possibles. Nous nous plaçons dans le cas de systèmes pour lesquels on dispose de 2 types de journaux :

- Un log des actions du système, tel qu'enregistré sur une API ou via une IHM ou un robot de test. Les événements enregistrés dans ce log le sont à une fréquence élevée qui peut être de l'ordre de la milliseconde ou de la seconde selon le niveau d'observation.

- Un échantillonnage régulier de métriques sur l'état global du système, à intervalle beaucoup plus espacé (de l'ordre de la dizaine de secondes à plusieurs minutes), analysant la consommation de ressources du système d'exploitation, telle que la consommation CPU, la consommation de mémoire, les défauts de page, les accès au réseau, etc.

L'absence de modèle formel du comportement est compensée par la présence de cette deuxième journalisation du système qui permet de retrouver des anomalies de fonctionnement à partir de l'impact sur les ressources utilisées par le système.

L'approche proposée par le projet pour la détection et la prédiction d'anomalies est illustrée sur la figure 3.

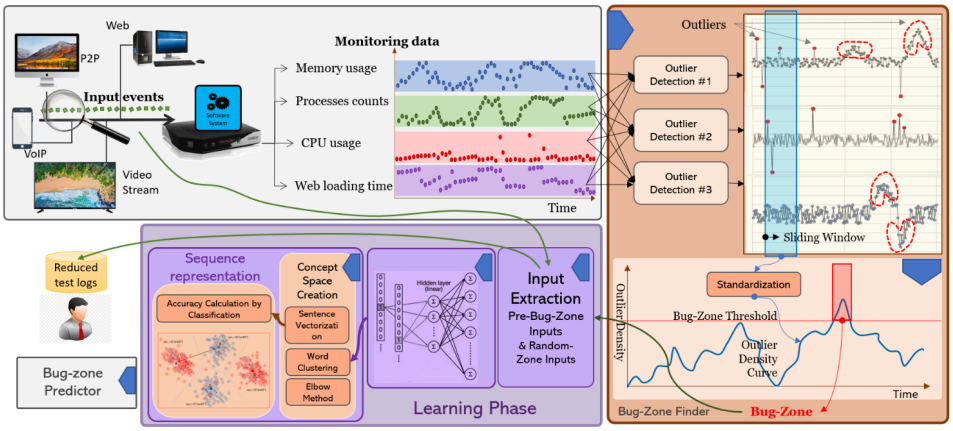


Figure 3: Détection et prédiction d'anomalie à partir de l'échantillonnage de l'état du système

Dans une première étape, des zones à risque (bug zone) sont automatiquement identifiées, par la présence corrélée de plusieurs métriques s'écartant de valeurs médianes pendant un certain intervalle (avec des seuils de détection qui peuvent être ajustés par les utilisateurs). Dans un deuxième temps, on va analyser les événements du journal des actions qui se sont produits en amont de la zone à risque. On représente les séquences d'événements comme des phrases d'un langage, et on va apprendre automatiquement à distinguer ces types de séquences de ceux de comportements normaux appris en dehors de zones à risque. On utilise pour cela des techniques également issues du TALN. L'identification des zones à risque réduit considérablement la charge des équipes de développement et validation pour le dépouillement des tests et des retours de production, et l'apprentissage automatique des séquences induisant des anomalies de comportement permet de prédire et potentiellement de prévenir les défaillances.

Sur l'étude de cas fournie par Orange, une première évaluation a permis de montrer qu'il était possible d'identifier et de prédire 70% des cas de réinitialisation du système, ce qui est proche du taux des réinitialisations sporadiques (liées à des pannes du système), sachant que les 30% restant semblent correspondre principalement sinon essentiellement à des réinitialisations forcées pour des raisons de gestion des équipements de tests.

## 5 Répertoire de code et de données en open-source Philae

Le projet PHILAE a vocation à mettre à disposition une boite à outil ”PHILAE Tool Box” en open source comme livrable de fin de projet. Il existe à ce jour 24 outils qui desservent 3 niveaux d'architecture au sein du projet PHILAE : 1. Outils de haut niveau dit ”Business Services”, 2. Outils de niveau médiant dit ”Builders” et finalement 3. Outils de bas niveau dit ”Helpers”.

Les outils permettent de mettre en place plusieurs techniques d'extraction des données (One-Hot encoding, Auto-encoding, Bag-of-words, ...), des techniques de clustering (Hidden Markov Model, MeanShift Clustering, ...), des techniques de réduction et d'évaluation de traces ainsi que la génération de scripts de test.

Les outils sont en ce moment en phase de migration d'un répertoire GitLab privé vers un répertoire publique GitHub accessible à cette adresse : `https://github.com/PHILAE-PROJECT`

Tous les outils seront disponibles avec des exemples fonctionnels en version finale du projet Philae en septembre 2022.

## 6 Conclusion

Le projet Philae a exploré différentes techniques d'analyse de logs pour extraire des traces de tests en vue de leur analyse et de la génération de scripts de tests avec des modèles d'apprentissage automatique. Les logs de test sont aussi analysés pour détecter des anomalies sur ces tests, permettant de les corriger ou de les optimiser.

Une boîte à outils open source permet de partager avec la communauté les différents composants développés, ainsi que deux cas exemple, avec les jeux de données associés, permettant la reproductibilité des résultats obtenus.

## References


[1] Bahareh Afshinpour, Roland Groz, Massih-Reza Amini, Yves Ledru, and Catherine Oriat. Reducing regression test suites using the word2vec natural language processing tool. In Bimlesh Wadhwa, Shailey Chawla, Benjamin Gan, Eng Lieh Ouh, Pornsiri Muenchaisri, Saurabh Tiwari, and Santosh Singh Rathore, editors, *Joint Proceedings of SEED & NLPaSE co-located with 27th Asia Pacific Software Engineering Conference 2020, Singapore [Virtual], December 1, 2020*, volume 2799 of *CEUR Workshop Proceedings*, pages 43–53. CEUR-WS.org, 2020.

[2] Elodie BERNARD, Julien BOTELLA, Fabrice AMBERT, Bruno LEGEARD, and Mark UTTING. Tool support for refactoring manual tests. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 332–342, 2020.

[3] Vahana Dorcie, Fabrice Bouquet, and Frédéric Dadeau. Clustering of usage traces for regression test cases selection. In *Artificial Intelligence in Software Testing (AIST'22)*, 2022.

[4] Mark Utting, Bruno Legeard, Frédéric Dadeau, Frédéric Tamagnan, and Fabrice Bouquet. Identifying and generating missing tests using machine learning on execution traces. In *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pages 83–90, 2020.