MBeeTle - un outil pour la génération de tests à-la-volée à l'aide de modèles *

Julien Lorrain[1]
julien.lorrain@femto-st.fr
Frédéric Dadeau[1]
frederic.dadeau@femto-st.fr
Elizabeta Fourneret[2]
elizabeta.fourneret@smartesting.com
Bruno Legeard[1,2]
bruno.legeard@femto-st.fr

[1]Institut FEMTO-ST, Besançon, France
[2]Smartesting Solutions & Services, Besançon, France

Résumé

Le Model-Based Testing est une activité permettant, à partir de la modélisation du système sous test (SUT), de générer des tests pour la validation du SUT par rapport aux spécifications de ce système. Cet article présente MBeeTle, un outil de génération de test à-la-volée pour le MBT, ainsi que ses principes et ces stratégies de génération. Il permet la recherche des anomalies profondes et la validation du modèle de test en générant et exécutant des pas de test sur la base d'une approche aléatoire. MBeeTle est présenté avec une expérimentation sur la une norme d'interfaces utilisateurs et logicielles pour les systèmes d'exploitation (POSIX).

1 Introduction et contexte

Dans ce papier nous présentons MBeeTle, un outil de génération de test à-la-volée [1] basé sur le Model-Based Testing (MBT) [2] pour l'exploration des systèmes sous test. Le MBT est une activité permettant de générer des tests abstraits à partir de la modélisation du SUT qui nécessite l'utilisation d'une couche d'adaptation afin de les concrétiser pour pouvoir les exécuter sur le SUT. La catégorie principale de génération de test est la génération dite hors-ligne, dont le principe est de générer des scénarios de test abstrait. Pour pouvoir les exécuter automatiquement sur le SUT, il faut créer le lien entre les données abstraites du modèle et le SUT. C'est ici que la couche d'adaptation est nécessaire pour associer à chaque donnée abstraite les informations de concrétisation permettant leur exécution sur le SUT. Une autre catégorie de génération, implémentée dans l'outil MBeeTle, est la génération à-la-volée. Le principe est d'attendre le résultat du premier pas de test avant de générer les pas suivants. Il est donc nécessaire de disposer de la couche d'adaptation pour mettre en œuvre cette méthode. Cette approche permet de générer des tests fonctionnels longs (*i.e.* plusieurs centaines de pas de test) basés sur les comportements modélisés du SUT.

*Cet outil a en partie été réalisé lors des projets ANR ASTRID OSEP et ANR ASTRID maturation MBT_SEC.

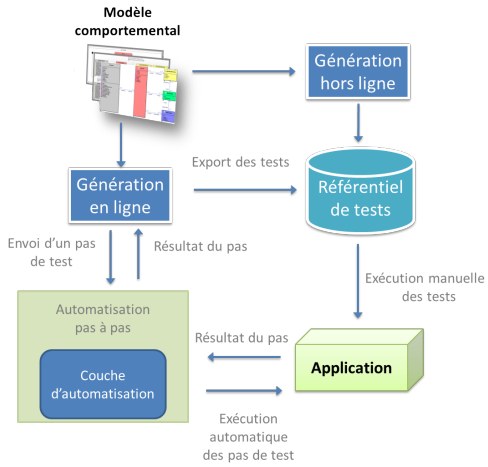

FIGURE 1 – Approche à-la-volée

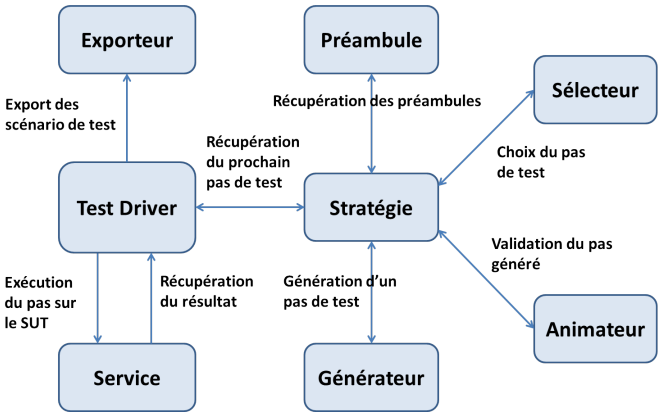

FIGURE 2 – Schéma de fonctionnement

MBeeTle est un outil de génération de tests à-la-volée qui génère puis exécute directement chaque pas de test sur le SUT. Il se base sur l'utilisation de modèles réalisés à l'aide de sous-ensembles des langages UML et OCL [3]. Il s'intègre dans une approche hors-ligne existante, représenté dans la figure 1 et réutilise des briques logicielles de l'outil Smartesting [1] CertifyIt [4] (un générateur de tests fonctionnels hors-ligne), notamment son animateur pour valider sur le modèle les différents pas de test générés par les algorithmes de MBeeTle.

2 L'outil MBeeTle

Le principe de MBeeTle est de générer des scénarios de test plus ou moins longs (plusieurs centaines de pas) à l'aide d'une combinaison d'algorithmes de génération de pas de test, de préambules et d'un ou plusieurs critères de sélection de tests nommés sélecteurs. Cette combinaison est appelé une *stratégie de génération*.

MBeeTle est composé de plusieurs modules, décrit dans la figure 2, ayant chacun un objectif dans la création d'un scénario de test à-la-volée. Le premier, le Test Driver, pilote et lie les autres modules. Les générateurs permettent, grâce à un tirage aléatoire, de générer des pas de test en se servant des informations du modèle. Les pas générés sont cohérents, *i.e.* utilisent des valeurs existantes dans le modèle, mais pas forcément valides par rapport à l'état courant du modèle. Un pas de test est valide si son animation sur le modèle ne rentre pas en contradiction avec les contraintes et/ou les pré-conditions présentes dans le modèle. C'est le rôle de l'animateur : si le pas est valide, il est envoyé au sélecteur. Si le sélecteur choisit le pas alors la stratégie le valide et le retourne au TestDriver qui, au travers du module de service, l'exécute et récupère le résultat de cette exécution. Cette suite d'actions est alors recommencée jusqu'à remplir les conditions d'arrêt de la campagne de génération à-la-volée.

L'interface utilisateur présenté figure 3 permet de configurer entièrement l'outil et d'afficher des informations de la campagne courante, comme le nombre de scénarios générés et la couverture des opérations et des comportements du modèle. Les deux algorithmes de génération de tests utilisés dans MBeeTle sont basés sur l'aléatoire. Le premier est purement pseudo-aléatoire et ne tient pas compte de ce qui a été généré précédemment. Afin de facili-

1. http ://www.smartesting.com/

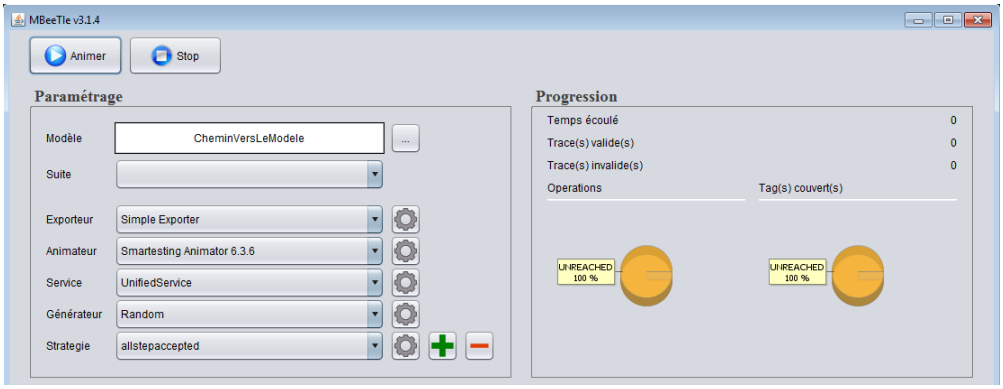

Figure 3 – L'interface de l'outil MBeeTle

ter le passage des comportements complexes, le second implémente une pondération sur les opérations et sur les paramètres.

Trois sélecteurs sont disponibles dans MBeeTle. Le premier est simple et utilise n'importe quel pas de test qui lui est soumis. Le second se base sur les comportements modélisés et optimise leur couverture en favorisant les pas de test qui augmentent la couverture des comportements. Le dernier explore les états du système dans des contextes plus divers que l'approche habituelle. Il se base sur les cas non-passants/passants, *i.e.* les pas qui provoquent un état d'erreur du système et les autres, le but étant de générer des séquences inhabituelles d'opérations qui peuvent provoquer de nouvelles anomalies.

Avant de lancer la génération l'outil peut importer des tests existants. Les bénéfices de cette ré-utilisation de tests sont multiples : augmentation de la diversité des contextes d'activation des comportements déjà testés en évitant les enchainements de comportements déjà couverts, diminution du temps de génération et passage des goulots d'étranglements du modèle (une difficulté inhérente aux approches aléatoire) et du coup augmentation de l'espace d'exploration. L'outil permet d'exporter les résultats sous différents formats.

3 Expérimentations

Le périmètre du standard POSIX [2] modélise la gestion de fichiers, d'utilisateurs et de droits pour les systèmes de type UNIX. Le modèle est composé de 19 classes, avec 24 opérations pour 2 442 lignes d'OCL et chaque interface de l'API est accessible à tout moment. Les tests hors-ligne ont été générés en 24 heures pour couvrir 88% du modèle. Nous avons choisi de faire des lancements/exécutions avec MBeeTle respectant un volume de 20 000 pas correspondant à celui des tests fonctionnels, tout en découpant les lancements en trois configurations types : 1000 scénarios de 20 pas de test, 250 scénarios de 80 pas de test et 100 scénarios de 200 pas de test. Pour chaque stratégie de génération on évalue la couverture des comportements, et le temps de génération de tests et d'exécution des pas de test. Par ailleurs nous avons généré, avec l'outil Pitest [3], 1290 mutants qui nous ont servit pour une analyse mutationelle. La génération hors-ligne tue 548 mutants avec 1484 tests, la génération à la volée en tue en moyenne 546 et la somme des deux approches en tue 630.

2. https ://standards.ieee.org/findstds/standard/1003.1-2008.html
3. http ://pitest.org/

De cette expérimentation ressort que MBeeTle permet de couvrir rapidement plus de la moitié des comportements du modèle (environ 16 minutes contre 3 heures pour la version hors-ligne). En réutilisation des tests hors-ligne générés, on augmente significativement la couverture du modèle. Par ailleurs MBeeTle est capable de générer rapidement un volume de pas de test conséquent (ici 20000 pas en 13,6 secondes en moyenne). De plus l'utilisation de cette approche a permis de détecter des mutants différents de ceux détectés par l'approche hors-ligne.

Par ailleurs, l'application de l'approche à-la-volée sur cette expérimentation montre que les artefacts de modélisation hors-ligne sont réutilisables avec un faible coût de main d'œuvre pour adapter le banc de test pour une utilisation pas à pas. Il en ressort également que MBeeTle peut être utilisé pour mettre au point le modèle en générant rapidement des scénarios de test.

4 Conclusion et perspectives

Nous avons présenté MBeeTle en association avec l'approche hors-ligne sur un modèle de taille réelle. Dans ce contexte il permet de compléter cette approche avec un faible coût d'adaptation et une réutilisation complète des ressources et des outils mis en place. Il permet d'augmenter l'espace d'exploration pour activer des comportements dans d'autres contextes. Par ailleurs sur un autre cas d'étude, une implémentation de la norme PKCS#11 [4], il a permis de montrer des erreurs dues a des fuites mémoires qui n'ont pas été trouvés par les outils classiques grâce à une utilisation intensive de l'outil [5].

Pour la suite, nous prévoyons d'adapter cette approche à-la-volée pour en faire une approche de tests en-ligne où nous utiliserons le retour du SUT pour calculer le pas de test pour calculer le prochain. De part son aspect modulaire et ses stratégies paramétrables, des expérimentations pour de la génération en-ligne (utilisation du retour d'exécution des pas de test pour décider du prochain pas de test à générer) [5] sont envisageables. Par ailleurs pour améliorer son efficacité, de nouvelles stratégies seront ajoutées pour palier aux difficultés rencontrées, par exemple sur les modèles à goulots d'étranglements.

Références

[1] J.-C. Fernandez, C. Jard, T. Jéron, and C. Viho, "Using on-the-fly verification techniques for the generation of test suites," in *Computer Aided Verification*, pp. 348–359, Springer, 1996.

[2] M. Utting and B. Legeard, *Practical model-based testing : a tools approach*. Morgan Kaufmann, 2010.

[3] F. Bouquet, C. Grandpierre, B. Legeard, F. Peureux, N. Vacelet, and M. Utting, "A subset of precise UML for model-based testing," in *Proceedings of the 3rd International Workshop on Advances in Model Based Testing*, pp. 95–104, 2007.

[4] F. Bouquet, C. Grandpierre, B. Legeard, and F. Peureux, "A test generation solution to automate software testing," in *Proceedings of the 3rd International Workshop on Automation of Software Test*, AST '08, (New York, NY, USA), pp. 45–48, ACM, 2008.

[5] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball, "Feedback-directed random test generation," in *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pp. 75–84, IEEE, 2007.

4. https ://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-11-cryptographic-token-interface-standard.htm

5. http ://ucaat.etsi.org/2015/presentations/FEMTOST_FOURNERET.pdf