Réutilisation d'artefacts projets pour le Model Based Testing

Hussein MHANNA[12], Hélène LE GUEN[1], Patrick LESERF[2], and Sébastien SAUDRAIS[2]

[1] prenom.nom@all4tec.net - ALL4TEC Laval, FRANCE
[2] prenom.nom@estaca.fr - ESTACA Laval, FRANCE

Résumé

Cet article présente une méthodologie pour faciliter l'emploi du Model Based Testing au sein de sociétés. Notre approche combine la génération de cas de test à partir des modèles de spécification / conception et la génération de cas de test à partir d'un modèle dédié aux tests. Elle consiste à utiliser des modèles de spécification / conception ou de test existants pour construire un modèle dédié aux tests. Un cas d'utilisation d'une machine ATM est présenté pour décrire le besoin et les problématiques associées. Un premier travail de transformation d'un ensemble de diagrammes de séquences en modèle d'usage est également proposé dans cet article. Ces travaux sont soutenus par le projet National Clarity [1], pour la transformation des diagrammes de séquences Capella [2] en modèle d'usage MaTeLo [3].
Mots clés : Model Based Testing, Ingénierie dirigée par les modèles, Transformation de modèles.

Abstract

In this paper, we present a methodology to facilitate the use of Model Based Testing in companies. Our approach combines test cases generation from specification / design models and test cases generation from a model dedicated to testing. It consists to use the model specification, design or existing tests to build a model dedicated to testing. A use case of an ATM machine is presented to describe the need and the associated issues. A first work of transforming a set of sequence diagrams to usage model is also presented in this article. This work is supported by the French project Clarity[1], for transforming Capella[2] Sequence diagrams into MaTeLo[3] usage model.
Keywords: Model Based Testing, Model Driven Engineering, Model Transformation.

1 Introduction

Dans le cycle de développement logiciel, le test est une des phases les plus coûteuses [12]. Les phases de tests représentent entre 30% et 50% du coût total du développement, et parfois même jusqu'à 70% [11]. Depuis ces dernières années, on observe que beaucoup de nouvelles solutions ont été mise en œuvres pour améliorer cette phase. Par exemple, il existe la séparation des équipes de développements et de tests utilisant des processus et des outils dédiés. L'automatisation de tests devient de plus en plus présente dans l'exécution des tests mais reste anecdotique pour la définition des tests à réaliser. Cette définition de test peut être réalisée par le Model Based Testing (MBT) qui consiste à générer automatiquement des tests de validation à partir d'un modèle décrivant certains aspects fonctionnels du système sous test (System Under Test : SUT) [14]. Il existe deux approches pour générer des cas de test à partir de modèle, la première consiste à les générer à partir des modèles de spécifications et/ou conception existants et la deuxième consiste à créer un nouveau modèle pour les générer.

1. http ://www.clarity-se.org
2. https ://www.polarsys.org/capella/
3. http ://www.all4tec.net/MaTeLo/homematelo.html

Le MBT a été utilisé avec succès sur de nombreux projets industriels et la plupart des études [2, 9] ont montré qu'il est un moyen efficace de détection de bugs du SUT, cependant il reste très peu utilisé dans le monde industriel. Les causes majeures de ce faible succès résident d'une part dans la difficulté à gérer et à maintenir un nouveau modèle dédié au test et d'autre part dans la nécessité d'un changement de processus de développement.

Nos travaux combinent les deux approches de MBT et proposent une nouvelle méthodologie pour le rendre plus facilement accessible aux sociétés. Notre objectif est de rendre utilisable l'existant, quelque soit le niveau de détail des modèles de spécification et de conception décrits au format tels que UML[1] et SysML[3] et des cas de test existants pour la construction d'un modèle de test, avec des interventions possibles de l'utilisateur au fil de la construction. Ce papier présente notre méthodologie permettant d'avoir un modèle exploitable pour d'éventuelles modifications manuelles, avec notamment la gestion d'une architecture hiérarchique de modèle. Un cas d'utilisation, distributeur de billets, illustre notre méthodologie et ses problématiques.

2 État de l'art et Motivations

Les deux approches principale du MBT utilisent différentes modélisation du systèmes (machine à états, systèmes de transitions étiquetées, diagrammes UML, réseaux de Petri, chaînes de Markov [5], etc). Toutes ces modélisations sont complémentaires et offrent les possibilités requises pour la modélisation et la génération des cas de test automatique [14].

La génération des cas de test à partir des modèles de spécifications et/ou conception existants est utilisée par [4, 8] permettant la génération des cas de tes à partir de diagrammes UML et SysML en utilisant à la fois des diagrammes statiques et comportementaux. Des données spécifiques pour le test sont écrites à l'aide d'expressions OCL ou ALF, afin de spécifier les pré/post conditions des opérations, pour calculer des tests de type boîte noire. Ces ajouts de données augmentent le niveau de détails des diagrammes qui ne sont pas nécessaires pour la spécification simple du système. Afin de générer des tests, la spécification doit être aussi complète que possible alors que généralement les concepteurs ne modélisent pas complètement leur système. En effet pour un système donné, souvent seuls les diagrammes de classes, quelques diagrammes de cas d'utilisations et quelques diagrammes de séquences sont conçus. Enfin, il n'est pas simple pour les ingénieurs de validation de connaître les informations nécessaires et utiles pour la génération des cas de test. Lorsque le modèle est trop bas niveau, faite par l'équipe de développement, il contient des informations inutiles pour la génération de test. Mais, lorsque le modèle est trop haut niveau, il contient trop peu d'informations pour pouvoir générer des cas de test exécutables et pertinents.

La génération des cas de test à partir d'un modèle dédié est utilisée par [6, 13]. Ce nouveau modèle est construit souvent manuellement à partir des exigences fonctionnelles ou d'un document de spécification logiciel. La validation du modèle est réalisée par l'examen minutieux des exigences du SUT afin d'en vérifier la consistance et la complétude. L'avantage de cette approche est d'avoir un modèle dédié aux tests bien conçu, modélisant les exigences fonctionnelles de système et contenant uniquement les données utiles et nécessaires pour pouvoir générer des cas de test pertinents et permettant de valider les exigences modélisées. D'autre part avec cette approche, il y a une indépendance entre le processus de développement et celui de test. L'inconvénient de ces travaux est la nécessité de créer et de maintenir manuellement ce nouveau modèle et donc d'effectuer un travail de modélisation parfois redondant avec celui réalisé en phase de spécification.

Notre approche combine ces deux approches pour construire automatiquement un modèle dédié exclusivement au test tout en réutilisant les artefacts déjà créés sur le projet.

3 Méthodologie

Notre méthodologie propose la création d'un modèle de test à partir d'un ensemble de modèle existant. Elle permet également la mise à jour des modèles de tests déjà créés lorsque les modèles de spécification / conception existants évoluent. Afin de pouvoir stocker les informations nécessaires pour créer le modèle d'usage, un méta-modèle *Lite Usage Model* (LUM) pour les modèles d'usages a été défini. Notre solution est composée de deux étapes (Fig. 1). La première est le passage des modèles existants, ensemble de modèles de spécification, de conception au format UML, SysML ou des cas de test ou autres formats, en méta-modèle LUM à l'aide de la transformation *Usage Model Transformer* (UMT). La deuxième étape permet la transformation d'un modèle au format LUM en modèle d'usage au format MaTeLo à l'aide de la transformation *MaTeLo Usage Model Transformer* (MUMT).

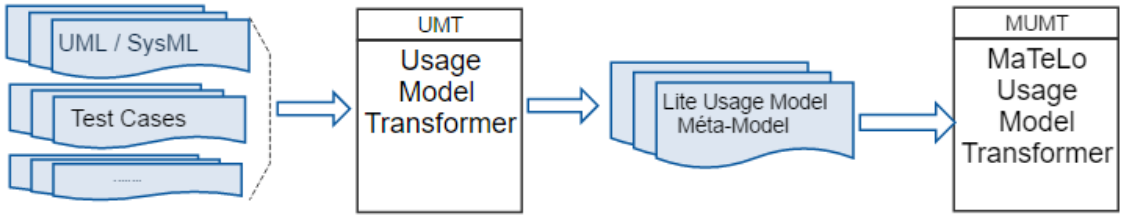

FIGURE 1 – Usage Model Transformer et MaTeLo Usage Model Transformer

4 Diagramme de séquence vers modèle d'usage

Pour illustrer l'intérêt et la complexité de nos travaux, nous nous sommes basés sur un exemple de diagrammes de séquence (Fig. 2) d'une machine ATM modélisant le retrait d'argent ainsi que le blocage d'une carte bancaire suite à trois saisies consécutives d'un code PIN erroné. Nous avons réalisé un premier algorithme 1 pour transformer les diagrammes de séquences en modèle d'usage. Dans cet algorithme, seuls les échanges entre l'utilisateur et le système (principe de test boîte noire), et de l'utilisateur avec lui même sont considérés. Un modèle d'usage est représenté sous la forme d'un graphe de test noté G et représenté par le tuple (S, s_0, s_n, V,

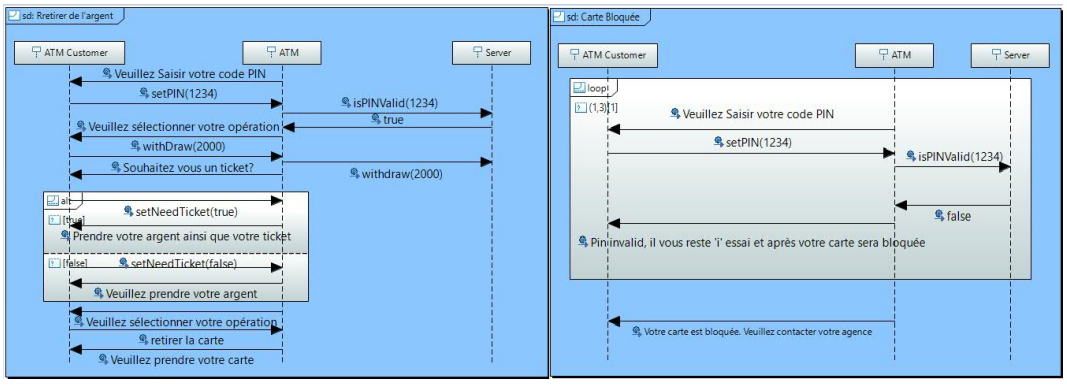

FIGURE 2 – Diagrammes de séquences d'une machine ATM

T), où S est un ensemble fini d'états, $s_0 \in S$ est un unique état de début dans la chaîne, $s_n \in S$ est un unique état de fin dans la chaîne, V est un ensemble des entrées/sorties et T est un ensemble de transitions. On définit une transition conditionnelle comme ayant un état source et plusieurs états successeurs, l'état successeur étant déterminé par la condition portée par la transition. L'objectif dans cet article étant de présenter la démarche de la transformation, nous ne détaillons pas ni tous les opérateurs ni la gestion des données. En appliquant cet algorithme sur les diagrammes de la figure 2, nous obtenons le modèle d'usage présenté dans la figure 3.

Algorithme 1 : Transformation de diagrammes de séquence en modèle d'usage

On considère G ={S, s_0, s_n, V, T}, S = {s_0, s_n} et SD l'ensemble des diagrammes à importer. L'état courant s_c est initialisé à s_0.
pour *chaque d dans SD* **faire**
  importContent(d) ; tr = createTransition() ;
  tr.setSource(s_c) ; tr.setTarget(s_n) ;
fin
procedure IMPORTCONTENT(d)
pour *chaque élément dans d* **faire**
  si *isFragment(elt) // si l'élément est un fragment* **alors**
    importFragment(elt) ;
  sinon si *isMessageOrOperation(elt) // si l'élément est un message ou opération* **alors**
    importMessage(elt) ;
fin

procedure IMPORTFRAGMENT(elt)
si *isAlt(elt)// si l'élément est un opérateur alt* **alors**
  createFirstChoiceTransition(elt) ; createSecondChoiceTransition(elt) ; importContent(elt) ;
sinon si *isLoop(elt)// si l'élément est un opérateur loop* **alors**
  createConditonalTransitional(elt) ;

procedure IMPORTMESSAGE(elt)
si *isElementExistInUsageModel(elt) // si l'élément existe déjà dans le modèle* **alors**
  si s_c *a une transition sortante ayant ce message/opération* **alors**
    s_c = l'état destination de cette transition ;
  sinon
    s_{source} = *existingState.getSource*(); tr = *createTransition*(*elt*);
    tr.setSource(s_c) ;tr.setTarget(s_{source});s_c = s_{source};
  fin
sinon
  tr = createTransition(elt) ; tr.setLabel(*le nom du message*) ;
  s_{new} = *createState*(); *tr.setTarget*(s_{new});s_c = s_{new};
fin

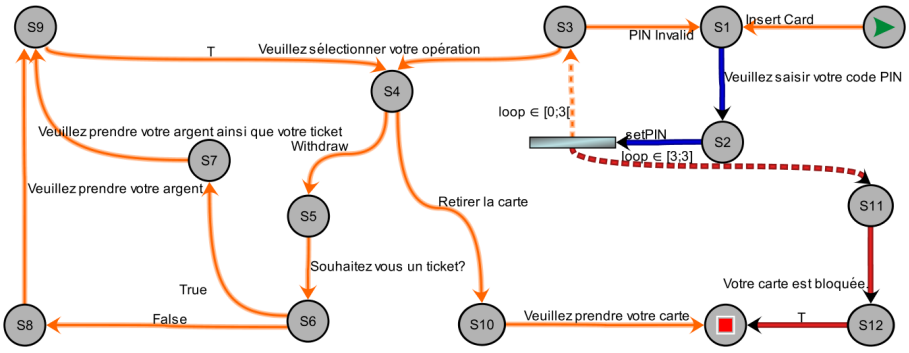

FIGURE 3 – Modele d'usage généré d'une machine ATM

A partir de cette transformation, on peut constater de manière non exhaustive que :

1. Souvent la totalité des comportements n'est pas décrite à l'aide des diagrammes de

séquences.

2. Il existe différentes possibilités de génération d'un modèle de test, par exemple pour le 'loop' dans la Figure 2, il peut être modélisé soit par trois transitions consécutives soit une boucle.
3. L'algorithme 1 ne prend pas en compte la construction d'une architecture hiérarchique permettant de factoriser des opérations commune à plusieurs tests. Cette amélioration permettrait de réduire la taille des modèles d'usage et est à prendre en compte.

A partir de ces constats, nous remarquons qu'une intervention manuelle est parfois nécessaire pour pouvoir choisir une solution parmi celles qui sont proposées.

5 Conclusion

L'utilisation de MBT permet d'optimiser la phase de test et ainsi réduire son coût. Cependant il est peu utilisé car de la création et la maintenance d'un nouveau modèle dédié au test est difficile. Dans cet article nous proposons une méthodologie permettant la génération automatique d'un modèle d'usage à partir des spécifications existantes tout en laissant la possibilité d'intervention du testeur. La difficulté de ces travaux sont multiples. On citera l'extraction des données utiles pour la construction d'un modèle d'usage et la proposition d'une architecture de modélisation efficace et satisfaisante. Nos travaux futurs consisteront, à extraire des données de différentes sources telles que UML, SysML et BPMN[10] ainsi que certains format de cas de test tel que ATX [4] ou TTCN [7]. Ils consisteront aussi à rendre possible la synchronisation et l'analyse d'impact sur le modèle d'usage lorsque les modèles d'entrées évoluent ou lorsque l'utilisateur modifie manuellement le modèle généré.

Références

[1] J. Dobing, B. PARSONS. How UML is used. Communications of the ACM, 2006.

[2] X. Luck E. Bernard, B. Legeard and F. Peureux. Generation of test sequences from formal specifications. Softw. Pract. Exper., 34(10) :915–948, August 2004, 2004.

[3] Moore Alan Friedenthal, Sanford and Rick Steiner. Practical guide to sysmlsystems modeling language. Morgan Kaufmann Publishers, Inc. : San Francisco, CA, 2008.

[4] C. GrandPierre. Stratégies de génération automatique de tests à partir de modèles comportementaux UML/OCL. ADU, 2008.

[5] H. Le Guen and T. Thelin. Practical experiences with statistical usage testing. In Software Technology and Engineering Practice, 2003. Eleventh Annual International Workshop on, pages 87–93,2003, 2003.

[6] Acquaroli B. Martelli A. Guiotto, A. Matelo : Automated testing suite for software validation. Proceedings of DASIA 2003 (ESA SP-532). 2-6 June 2003, Prague, Czech Republic, 2003.

[7] C. Willcock J. Grabowski, A. Wiles and D. Hogrefe. On the design of the new testing language ttcn-3, 2000.

[8] J. Lasalle. Génération automatique de tests à partir de modèles sysml pour la validation fonctionnelle de systèmes embarqués. Université de Franche-Comté,2012. NNT : 2012BESA2015. tel-00762053v2, 2012.

[9] S. Jell M. Sarma, P. V. R. Murthy and A. Ulrich. Model-based testing in industry : A case study with two MBT tools. In Proceedings of the 5th Workshop on Automation of Software Test, AST '10, pages 87–90, New York, NY, USA, 2010. ACM., 2010.

4. https ://wiki.asam.net/display/STANDARDS/ASAM+ATX

[10] Recker. BPMN modeling - who, where, how and why. BPTrends, 2008.

[11] M. Rozenberg. Test logiciel. Eyrolles, 1998.

[12] K. Sayre. Improved techniques for software testing based on markov chain usage models. PhD thesis, University of Tennessee, Knoxville, 1999.

[13] J. Tretmans and E. Brinksma. Torx : Automated model-based testing. Côte de Resyste, 2003.

[14] M. Utting and B. Legeard. Practical model-based testing. Morgan Kaufmann, 2007.