VETESS: IDM, Test et SysML

Frédéric Fondement[1], Pierre-Alain Muller[1], Brice Wittmann[1],
Fabrice Ambert[2], Fabrice Bouquet[2], Jonathan Lasalle[2], Émilie Oudot[2], Fabien Peureux[2],
Bruno Legeard[3], Marc Alter[4] et Claude Scherrer[4]

[1]Université de Haute Alsace, Mulhouse, France
[2] Université de Franche-Comté, Besançon, France
[3] Smartesting, Besançon, France
[4] Clemessy, Mulhouse, France

Résumé : Il apparaît souvent que les processus d'ingénierie système sont en fait décomposés en phases discontinues où trop peu d'informations sont partagées entre les différentes équipes, par exemple entre les équipes de design et de tests. Cette faiblesse peut être palliée par l'utilisation de modèles de spécifications qui jouent alors le rôle de référentiel pour l'ensemble des équipes participant au cycle de vie du logiciel. Ce type de modèle est couramment utilisé comme base dans les activités de conception, de vérification, ou encore de test. Le test basé sur les modèles est une approche originale où sont automatiquement générés des cas de test et des scripts de test exécutables à partir d'une spécification du système sous test. Cette spécification prend la forme d'un modèle comportemental, permettant ainsi au générateur de tests de déterminer, d'une part, quels sont les contextes d'exécution pertinents et, d'autre part, de prédire les effets sur le système de ces exécutions. Le but du projet VETESS est de rendre possible cette approche pour valider les systèmes embarqués automobiles. Il s'agit ainsi de mettre en œuvre et d'outiller un processus automatique permettant de dériver, d'un modèle de spécification décrit avec un sous-ensemble du langage de modélisation SysML, des cas de test, et de produire ensuite les scripts de test correspondants à exécuter sur banc de test automobiles.

Mots-clefs : Test Basé sur les Modèles (MBT), Ingénierie Dirigée par les Modèles (IDM), SysML.

1. INTRODUCTION

Très souvent, la démarche de conception de systèmes est principalement discontinue, basée sur l'utilisation de documents et d'artefacts intermédiaires constituants des vues différentes et dont la mise en cohérence constitue une difficulté considérable. La pratique de la modélisation pour la conception système a émergé depuis plusieurs années, notamment depuis l'apparition de langages comme StateMate [5], UML [7] ou SysML [6]. Les industriels du secteur ont ainsi acquis un savoir-faire dans le domaine et utilisent couramment les techniques de modélisation en phase de conception. D'autre part, les techniques de génération de tests à partir de modèles, plus communément appelées MBT (Model-Based Testing) [9], ont connu depuis plusieurs années un intérêt et un essor considérables. Cette attractivité, de la part du milieu scientifique comme industriel, s'explique notamment par la complexité toujours croissante des systèmes informatisés et le besoin récurrent de garantir une qualité de service et une fiabilité optimales. En effet, générer des tests à partir de modèles de spécifications permet d'assurer un niveau de confiance élevé et de fournir des éléments de mesure rationnels vis-à-vis du respect des propriétés énoncées dans le cahier des charges de l'application à tester.

L'objectif du projet VETESS [10] est de combler le fossé entre spécification et vérification en fondant la génération automatique de tests sur des modèles d'ingénierie systèmes en UML/SysML et en les exécutant sur bancs de test. Plus précisément, il s'agit de déduire d'un modèle de spécification comportemental d'un système, tel qu'il pourrait être produit en phase de conception, un ensemble de tests exécutables sur le système réalisé. Pour atteindre cet objectif, le projet se fonde sur les technologies des produits commerciaux Test Designer™ (de Smartesting [8]), un générateur de tests basé sur les modèles UML/OCL, et TestInView (de Clemessy [3]) qui permet de gérer et d'exécuter des jeux de test sur des modèles de simulation ou sur des bancs de test mécatroniques. Cet article présente la démarche proposée dans le cadre du projet VETESS et décrit la chaîne outillée qui permet son automatisation.

Le reste de cet article est organisé comme suit. Dans une première partie, sont introduits les principes généraux du test basé sur les modèles. En section 2 seront décrits plus avant les composants de la chaîne outillée. Avant de conclure, un cas d'étude, portant sur un module de gestion de l'essuyage avant d'un véhicule, est développé en section 3.

2. LE TEST BASÉ SUR LES MODÈLES

Le test basé sur les modèles, ou Model-Based Testing (MBT) [9], permet de générer un ensemble de cas de test à partir d'un modèle fonctionnel spécifiant le comportement du système sous test (SUT – System Under Test) ainsi que celui de son environnement. Ce modèle décrit donc ce qu'on est en droit d'appeler le contrat du système à tester [2]. Ces cas de test permettent de valider la cohérence comportementale du système sous test en comparant dos-à-dos les réactions observées sur le système développé avec celles décrites dans la spécification (qui jouent alors le rôle d'attendus).
Le principe général est le suivant. Tout d'abord, l'outil de génération de tests va explorer le modèle de spécification pour en extraire des objectifs de test afin de satisfaire un critère de couverture du modèle. Dans ce contexte, les critères les plus communément utilisés consistent à définir les éléments du modèle que l'ont veut voir couverts par le jeu de test : couvrir, par exemple, tous les états ou toutes les transitions spécifiées dans le modèle. Bien évidemment, la nature du critère influe directement sur le nombre de tests générés et sur leur pertinence vis-à-vis du domaine d'application du système sous test.
Ensuite, pour chacun de ces objectifs, un générateur va déterminer la séquence de stimuli à exercer sur le système pour couvrir cet objectif. Finalement, pour chacune de ces séquences, le générateur va simuler les effets de l'exécution de la séquence considérée sur le modèle de spécification afin de définir les valeurs attendues lors de l'exécution de cette même séquence sur le système sous test. Bien entendu, le modèle comportemental doit être exécutable, donc le langage dans lequel il est exprimé doit définir une sémantique opérationnelle claire. Par exemple, pour UML, une stimulation correspondra à un appel d'opération, et les effets seront définis par les mises à jour de propriétés et/ou une évolution des machines à états. Chacune de ces séquences de stimulations (stimuli), renseignée des effets attendus (points de contrôle), donnera lieu à un test dit abstrait, car décrit avec le même vocabulaire et à la même abstraction que le modèle de spécification dont il est issu.
Pour pouvoir exécuter les tests abstraits sur le système réel, ces tests doivent être concrétisés. Cette tâche consiste à mettre en correspondance les stimuli et les points de contrôle avec la réalité « physique » du système sous test. Il faut ainsi décrire comment envoyer chacun des messages au système et comment contrôler son état. Si on considère un modèle comportemental décrit en UML, il faut donc indiquer comment invoquer les opérations du système sous test (SUT – par exemple par des envois de messages sur bus CAN), et comment vérifier la valeur d'une propriété, et/ou l'état actif d'une machine à états (avec les problèmes d'observabilités des tests classiques [4]).
Le principal avantage du MBT réside dans sa capacité à augmenter la productivité durant la phase de test (générer automatiquement des tests accompagnés des résultats attendus permet un traitement et une exécution plus rapide), et à maîtriser la qualité et la pertinence des tests en garantissant une couverture donnée des fonctionnalités et des combinaisons de stimuli testées. Cependant, si cette technique permet de maîtriser la couverture des exigences fonctionnelles spécifiées dans le modèle comportemental du système à tester, elle se limite aussi à cet aspect : ce qui n'est pas modélisé ne sera donc pas testé. C'est pourquoi, en pratique, un modèle de spécification est souvent réalisé par rapport aux fonctionnalités qu'on désire tester ; on ne se trouve donc plus forcément en présence de modèles directement issus de l'analyse, mais de modèles dédiés à la génération de tests (ces modèles de tests peuvent toutefois découler des modèles d'analyse).

3. CHAÎNE OUTILLÉE

Deux buts principaux sont poursuivis par le projet VETESS : l'utilisation du langage SysML (complété de contraintes OCL [11]) pour la description du modèle comportemental du système sous test pour une génération automatique de tests abstraits, et la concrétisation de ces tests générés sur la plate-forme abstraite d'exécution de tests TestInView. La génération des tests, sur la base du modèle SysML, s'appuie sur l'outil Test Designer™, qui a initialement été conçu pour accepter des modèles comportementaux décrits dans un dialecte UML.

3.1. Test Designer™

C'est sur le principe du MBT que fonctionne l'outil Test Designer™ (TD), logiciel édité par la société Smartesting. TD est un générateur de tests basé sur les modèles qui prend en entrée un modèle de spécification, exprimé dans un sous-ensemble du langage UML (UML4MBT), qui spécifie le comportement du système à tester. TD permet alors d'interpréter ce modèle UML et d'automatiser, par application de stratégies définies, la génération des cas de test et des résultats attendus. Les tests générés prennent ainsi la forme de séquences d'appels d'opération et/ou d'envois de signaux (stimuli) agrémentées des résultats attendus sur le système (observation).

Afin d'accepter les ateliers de modélisation les plus courants du marché, des outils ont été développés pour construire des modèles de type UML4MBT à partir de modèles exprimés dans le langage UML standard. En effet, UML est un langage jouissant d'une certaine popularité qui permet d'exprimer le comportement ainsi que l'environnement d'un système. Ces outils ont été intégrés à des modeleurs du marché afin de permettre la génération de modèles UML4MBT directement depuis les environnements de développement de modèles UML.

3.2. TestInView

La plate-forme TestInView (TIV), éditée par la société Clemessy, est un environnement logiciel pour l'automatisation de tests sur des systèmes électroniques embarqués. Une séquence de test TestInView peut être exécutée de deux manières: soit en mode simulé, où le système sous test est en fait un modèle d'exécution mathématique (Matlab, Labview voire un programme .NET), soit sur banc de test mécatronique. Une caractéristique du système TestInView réside dans le fait que la description des séquences de tests ne dépend pas du mode d'exécution.

Sur ce principe, le système sous test est décomposé en une série de variables évoluant en continu au cours du temps. Nous nommerons cette évolution au cours du temps signal, dont la nature est continue mais peut toutefois être échantillonnée. Il est cependant important de noter que cette notion de signal n'a aucun rapport avec la notion de signal SysML, qui correspond, elle, à un envoi de données asynchrone. Ces variables représentent soit les entrées du système (comme une commande ou un capteur), soit une observation sur le système (son état ou ses sorties).

Une séquence de test décrit les signaux à envoyer sur les variables d'entrée, et les signaux attendus sur les variables de sortie. La validation d'un test est contrôlée via des comparateurs intelligents permettant certains types de variations sur les signaux de sorties mesurés par rapport aux signaux de sorties attendus (admission de seuil, filtres passe-bas, etc.).

3.3. Approche générale

L'approche générale est résumée par la figure 1. TD ① peut être vu comme un composant ou une transformation de modèle prenant en entrée un modèle comportemental dans un dialecte UML (UML4MBT ❶), et produisant un ensemble de scenarii de test abstraits sous la forme d'envoi de messages et de contrôles d'états ❷.

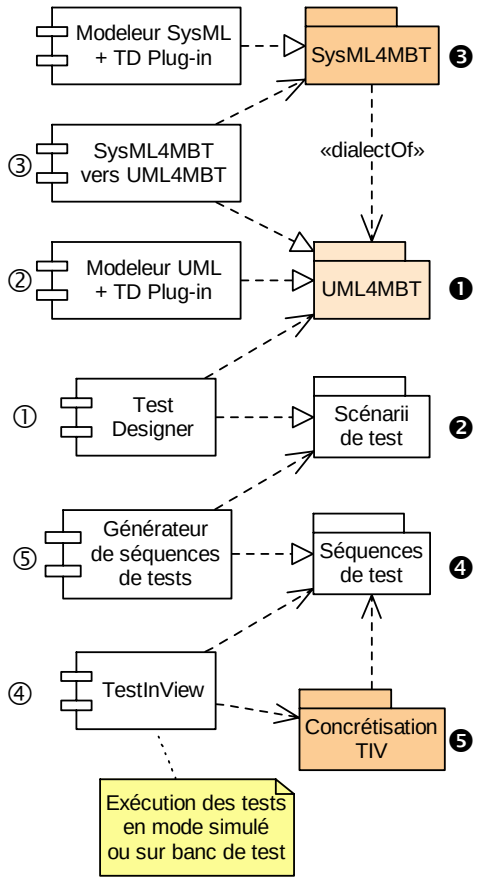

Figure 1 : Génération de séquences de test à partir de modèles SysML

Dans la figure 1, le métamodèle UML4MBT joue donc le rôle d'interface requise du point de vue de TD. Des plugins intégrés aux modeleurs UML (comme Topcased ou Rational Software Modeler) permettent de fournir, depuis tout modèle UML, les modèles UML4MBT ②.

Afin de prendre en compte un sous-ensemble du langage SysML (en remplacement d'UML4MBT) pour la définition des modèles comportementaux, a été développé le métamodèle SysML4MBT ❸ précisant quels concepts sont véritablement pris en compte pour la génération de tests.

Pour assurer la cohérence et suivre les évolutions d'UML4MBT, le métamodèle SysML4MBT a été développé comme un dialecte d'UML4MBT, c'est-à-dire UML4MBT auquel des concepts ont pu être retirés ou ajoutés. Cette technique permet de garantir une certaine proximité entre UML4MBT et SysML4MBT. Cette proximité a permis de simplifier le développement de la transformation qui consiste à produire, à partir d'un modèle SysML4MBT, un modèle de comportement équivalent exprimé dans le langage UML4MBT ③: seuls les concepts ajoutés à UML4MBT pour l'obtention de SysML4MBT ont du être pris en compte lors de l'implémentation de la transformation, les autres étant traduits de manière purement automatique.

La plate-forme TestInView ④, quant à elle, est capable d'une part d'exécuter une séquence de tests ❹ donnée sous la forme d'envoi de séquences de valeurs et, d'autre part, de contrôler l'évolution dans le temps des sorties. Comme ce dernier outil offre une abstraction sur le système à tester (qui peut être indifféremment un modèle mathématique, un banc de test, un programme de simulation, etc.), il peut être considéré comme une plate-forme abstraite d'exécution de tests [1], ciblant indifféremment des systèmes simulés ou des bancs de test.

Les tests générés par Test Designer sont des séquences de test abstraites qu'il est nécessaire de concrétiser (cf. §2 et 3.1). Cette concrétisation est rendue automatique par une génération de séquences de test ⑤. Cependant, les scenarii de test ❷ et les séquences de test ❹ sont de natures très différentes: les premiers sont de nature discrète et sont basés sur des événements de type envoi de message, alors que les seconds sont de nature continue en se basant sur la notion de signaux continus (tout de même échantillonnés). Ce composant est donc en charge de traduire des modèles discrets en interactions continues.

Sur le principe, l'évolution des arguments, envoyés par les messages dans les scenarii de test, est traduite en signaux. Les signaux attendus des variables de sortie ou d'observation peuvent eux être générés de deux façons. Soit ils sont calculés en fonction des observations générées par TD, soit ils sont obtenus par capture en exécutant les séquences de test en mode simulé. Dans ce dernier cas, les signaux obtenus par ce bais devront correspondre aux signaux obtenus sur le banc de test.

Pour conclure, cette chaîne outillée s'appuie sur deux éléments à fournir : un modèle comportemental décrit en SysML ❸ et une concrétisation indiquant à quoi correspondent, sur le banc de test, les paramètres des opérations et des signaux SysML décrits dans le modèle comportemental ❺. En plus de cela, un simulateur exposant le comportement continu non décrit dans le modèle SysML4MBT peut permettre d'affiner les verdicts en fournissant les signaux continus attendus sur les sorties, en rapport avec chacun des tests générés par la chaîne outillée.

4. CAS D'ÉTUDE : FONCTION ESSUYAGE AVANT

Afin de démontrer, sur un exemple réel, la faisabilité technique de la démarche proposée dans le cadre du projet VETESS, la chaîne outillée a été mise en œuvre sur une fonction d'essuyage avant d'un véhicule. Cette fonction Essuyage Avant, concernant un système fortement discret conditionné par les événements issus principalement de la manipulation du commodo du véhicule, permet de contrôler l'essuyage du pare-brise en mode manuel (vitesse lente, rapide ou intermittente), l'essuyage automatique avec capteur de pluie et les actions des essuie-glaces lors du lavage du pare-brise. La démarche outillée mise en œuvre sur ce cas d'étude, illustrée par la figure 2, se décompose en quatre étapes :

1. **Modélisation** : Ce cas d'étude a donné lieu à la réalisation d'un modèle SysML comportemental basé sur une représentation physique de l'élément sous test (fonction essuyage avant d'un véhicule) et sur la représentation de son environnement qui modélise l'ensemble des stimuli. Le modèle SysML de ce système est constitué par un diagramme de blocs, des diagrammes de blocs internes et des diagrammes d'états-transitions.

   Le diagramme de blocs permet de modéliser l'ensemble des composants physiques entrant en jeu dans la fonction Essuyage Avant. Le module de gestion de la fonction Essuyage Avant est ainsi modélisé par un bloc contenant trois autres blocs modélisant respectivement le module de gestion de l'essuyage avant manuel et l'essuyage avant automatique, le module du mode lavage avant, et finalement le module de gestion des priorités d'essuyage. L'environnement de la fonction Essuyage Avant est, quant à lui, représenté dans ce diagramme par un bloc contenant autant de blocs que d'éléments faisant partie de l'environnement (fonction d'allumage du véhicule, commodo d'essuie-glaces, liaison série avec le capteur de pluie,…).

La communication entre la fonction Essuyage Avant et son environnement est en effet effectuée via des signaux. Des signaux internes existent également au sein de la fonction, pour la communication entre les modules. La description de ces communications, que ce soient les communications avec son environnement ou les communications internes, est modélisée au travers de diagrammes de blocs internes.

Finalement, le comportement dynamique de chacun des éléments de la fonction Essuyage Avant et de son environnement est modélisé à l'aide de diagrammes états-transitions complétés de contraintes OCL afin de spécifier formellement les modifications induites par la réception des stimuli.

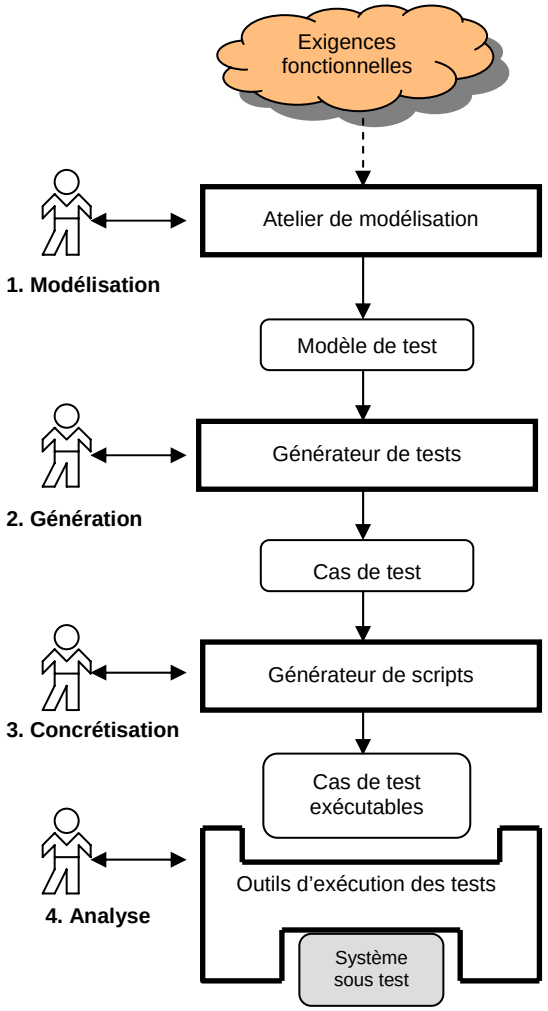

Figure 2 : Processus de génération des tests avec la chaîne outillée VETESS

2. **Génération** : Tester cette fonction consiste à dériver, du modèle SysML, des séquences de stimuli, et à vérifier que le comportement réel de l'élément sous test (un modèle exécutable de simulation) est fidèle au comportement spécifié dans le modèle SysML. La phase de génération des cas de test abstraits a donc été réalisée sur la base du générateur de test TD, dont les capacités ont été étendues pour prendre en entrée des modèles SysML et d'être en mesure d'appliquer des critères de couverture dédiés à ce type de formalisme. L'application d'un critère de type « couverture de toutes les transitions » sur les diagrammes états-transitions SysML a engendré sur ce cas d'étude la génération automatique de près de 150 cas de test abstraits.

3. **Concrétisation** : De manière automatique, un générateur de scripts transforme les tests abstraits générés par TD en scripts de tests exécutables par TestInView. Suite à cette génération de scripts, une couche de concrétisation, réalisée manuellement, donne la correspondance entre les invocations d'opérations abstraites, présentes dans les tests abstraits, et les actions réelles à réaliser sur le système sous test. Ces deux opérations successives permettent de produire des cas de test sous une forme exécutable.

4. **Exécution et Analyse** : À la suite de la phase de concrétisation, les cas de tests peuvent être directement exécutés par TestInView sur le système sous test. Dans le cadre de cette étude, le système sous test mis en œuvre consistait en un modèle exécutable de simulation écrit en langage .NET. Cette phase, menée avec succès, a permis de vérifier que le comportement réel de l'élément sous test était fidèle au comportement spécifié dans le modèle SysML initial sur l'ensemble des cas de test généré.

Ce cas d'étude a ainsi permis une première validation de la méthode définie au sein du projet VETESS et une première mise en œuvre de la chaîne outillée associée de bout en bout. Un second cas d'étude, portant sur un module de gestion de colonne de direction automobile est aujourd'hui en cours de réalisation. Cette seconde expérimentation permettra d'évaluer la méthode sur une application s'appuyant davantage sur un processus continu puisque l'état de la colonne de direction est continuellement évalué en fonction de la configuration de son environnement (caractéristiques de la route et de la conduite du véhicule).

5. CONCLUSION

Le projet VETESS est un projet de R&D collaboratif focalisé sur l'élaboration d'une méthode outillée de génération automatique de scripts de test pour les systèmes embarqués dans le domaine automobile. Il s'inscrit ainsi pleinement dans le contexte de la montée en puissance de l'ingénierie dirigée par les modèles pour la validation de systèmes embarqués dans les véhicules.

La méthode proposée dans le cadre de ce projet consiste à mettre en œuvre des techniques issues de la génération de tests à partir de modèle. Plus précisément, il s'agit, à partir d'un modèle de spécification du système sous test, écrit en langage SysML, d'élaborer une chaîne automatisée permettant d'engendrer des cas de test abstraits, puis de les concrétiser pour permettre leur exécution directement sur les bancs de test.

Un premier cas d'étude, concernant un module de gestion de l'essuyage avant d'un véhicule automobile, a permis de valider l'approche méthodologique proposée. Une chaîne outillée est aujourd'hui opérationnelle, et sa mise en œuvre, sur ce cas d'étude, a montré la faisabilité de l'automatisation du processus de génération et d'exécution des tests sur simulateur. La prise en charge d'un second cas d'étude, portant sur un système de gestion de colonne de direction, est en cours de réalisation, et permettra de consolider la démarche proposée dans un contexte de système fortement continu, et d'améliorer, dans ce contexte, les capacités de la chaîne outillée existante.

RÉFÉRENCES

[1] J. P. Almeida, R. Dijkman, M. van Sinderen, L. Ferreira Pires : *On the notion of abstract platform in MDA development* ; Eighth IEEE International Enterprise Distributed Object Computing Conference, EDOC, 20-24 septembre 2004, Monterey, Californie, États'Unis, 2004.

[2] A. Beugnard, J.-M. Jézéquel, N. Plouzeau et D. Watkins : *Making components contract aware* ; Computer 32 (1999), n° 7, pp. 38-45.

[3] The Clemessy web site, http://www.clemessy.fr, 2010.

[4] R.S. Freedman : *Testability of software component* ; IEEE Transactions on Software Engineering, vol. 17, n° 6, juin 1991.

[5] D. Harel : *Modeling reactive systems with Statecharts* ; Mac Graw Hill, 1998. ISBN 0 070 26205 5.

[6] Object Management Group. Systems Modeling Language (SysML), version 1.1, OMG Document formal/2008-11-01, novembre 2008.

[7] Object Management Group, Unified Modeling Language (UML) Superstructure, version 2.2, OMG Document formal/2009-02-02, février 2009.

[8] The Smartesting web site, *http://www.smartesting.com*, 2010.

[9] M. Utting et B. Legeard. *Practical Model-Based Testing – A tools approach* ; Elsevier Science, 2006. ISBN 0 12 372501 1.

[10] VETESS, site du projet : http://lifc.univ-fcomte.fr/vetess, 2010.

[11] J. Warmer et A. Kleppe : *The Object Constraint Language: Precise Modeling with UML* ; Addison-Wesley, 1996. ISBN 0 201 37940 6.