Intégration de TOBIAS et UCASTING pour la génération de tests

O. Maury, Y. Ledru, L. du Bousquet

LSR-IMAG
BP 72, 38402 St Martin d'Hères cedex, France
Tel : +33 4 76 82 72 14, Fax : +33 4 76 82 72 87
{Olivier.Maury, Yves.Ledru, Lydie.du-Bousquet}@imag.fr

Résumé : Nous présentons une approche basée sur la combinaison de deux techniques de génération de tests. D'une part, l'outil TOBIAS permet de générer de manière systématique de nombreuses séquences de test à partir d'une description abstraite. D'autre part UCASTING permet de générer des données de test à partir de la spécification exprimée en termes de pré/post-conditions. Nous étudions trois façons d'intégrer UCASTING avec TOBIAS. La première vérifie la conformité de séquences de test générées par TOBIAS. La deuxième calcule les valeurs des paramètres de séquences partiellement dépliées. La troisième précise cette recherche par l'ajout de contraintes.

Mots clés : test de conformité, sélection de valeurs de tests, test combinatoire,

Abstract: TOBIAS is a tool, which unfolds abstract test sequence descriptions into executable test cases. UCASTING is a conformance testing tool that selects test data from a specification. In this article we study 3 types of connection between both tools.

Keywords: Conformance testing, test data selection, combinatorial testing

1. INTRODUCTION

Le test de logiciel est aujourd'hui l'une des techniques les plus utilisées pour les activités de validation et de vérification. Cet article s'intègre dans le cadre du *test de conformité* [1], qui suppose l'existence d'une spécification du logiciel à tester.

- Elle constitue la référence à laquelle le comportement de l'implantation doit être comparé. Elle constitue un oracle et permet ainsi de décider de la réussite ou de l'échec de chaque test.
- Elle décrit l'ensemble des comportements admissibles et permet de définir des notions de couverture. Il est alors possible d'évaluer la pertinence d'un jeu de tests.
- Elle peut constituer le point de départ de la génération, manuelle ou automatique, des jeux de tests. Il existe ainsi plusieurs outils de génération de test de conformité. Par exemple, les outils TGV [8] et TorX [13] exploitent des spécifications comportementales. Les outils CASTING [14] et BZ-TT [10] s'appuient sur des spécifications décrites sous forme de pré/post-conditions.

Au cours du projet RNTL COTE[1], notre équipe a développé l'outil TOBIAS [2], dont le principe a été élaboré à partir de constatations expérimentales [4] : la validation d'une application industrielle nécessite de nombreux tests, et nombre d'entre eux ne diffèrent que par les valeurs de leurs paramètres, les instances auxquelles ils s'appliquent, ou par le nombre d'appels consécutifs à une méthode ou un groupe de méthodes. Dans TOBIAS, l'utilisateur peut exploiter ces similarités pour produire un ensemble de tests à partir d'une description abstraite, dite « schéma de test ». Le mécanisme d'abstraction offert par ces schémas permet au testeur de gérer facilement la taille du jeu de tests. La pertinence du jeu de test reste conditionnée par la qualité des schémas de test, i.e. par l'expérience et le travail d'analyse du testeur.

Dans cet article, nous étudions la combinaison de TOBIAS avec l'outil UCASTING [15] (issu aussi de COTE). L'objectif de cette intégration est d'assister le testeur dans la sélection des données de test. La section 2 présente une étude de cas illustrant notre démarche. La section 3 introduit et évalue l'outil TOBIAS. Les sections 4 et 5 proposent et évaluent trois types de connexion entre TOBIAS et UCASTING. Enfin, la section 6 résume le travail réalisé et en tire les conclusions.

2. UNE ETUDE DE CAS

Soit un système élémentaire de gestion de tampons, dont la spécification VDM [9,5] est donnée Fig. 1.
Le système est composé de trois tampons (`b1`, `b2` et `b3`). Les tampons sont représentés par une valeur entière (supérieure ou égale à 0) qui caractérise le nombre d'éléments stockés dans le tampon. La taille maximale supportée par le système est de 40 éléments. Le système doit répartir les éléments entre les trois tampons tels que `b1` soit plus petit que `b2`, lui même que `b3` et qu'il y ait au plus 15 éléments d'écart entre `b1` et `b3`. Ceci est caractérisé par l'invariant donné ligne 7 Fig. 1.
L'état du système est modifié par les opérations :

- `Initialiser` pour initialiser les tampons à zéro.
- `Ajouter` qui prend en paramètre un entier strictement positif représentant le nombre d'éléments à rajouter dans les tampons. Cette opération doit répartir la charge entre les éléments de sorte que l'invariant soit respecté.
- `Enlever` qui prend en paramètre un entier strictement positif représentant le nombre d'éléments qui sont retirés des tampons.

La spécification des opérations est définie en termes de pré et post-conditions. Les identificateurs marqués par un ~ dans la post-condition correspondent à la valeur de ces variables dans l'état initial.
La spécification est non-déterministe. Par exemple, elle n'impose pas la façon dont doivent être modifiés les tampons en cas d'ajout ou de retrait d'éléments.

[1] COTE a rassemblé des équipes de Softeam, de France Télécom R&D, de Gemplus, de l'IRISA et du LSR, autour du test de conformité de composants logiciels (2000-2002).

```
1.    state buffers of
            b1 : nat
            b2 : nat
            b3 : nat
5.
      inv mk_buffers(b1,b2,b3) ==
            b1+b2+b3<=40 and 0<=b1 and b1<=b2 and b2<=b3 and b3-b1<=15

      init B == B = mk_buffers(0,0,0)
10.   end

      Initialiser: ()==>()
      Initialiser() ==
      pre true
15.   post b1+b2+b3=0

      Ajouter: nat ==> ()
      Ajouter(nb) ==
      pre nb>0 and nb<=5 and b1+b2+b3+nb<=40
20.   post b1+b2+b3 = b1~+b2~+b3~+nb

      Enlever: nat ==> ()
      Enlever(nb) ==
      pre nb>0 and nb<=5 and nb<=b1+b2+b3
25.   post b1+b2+b3 = b1~+b2~+b3~-nb
```

Fig. 1 Une spécification VDM du système des tampons.

3. TOBIAS

3-1. Le principe

Plusieurs expériences industrielles, dont [4], ont montré que les tests d'une même suite de tests présentent de nombreuses similarités. Beaucoup sont identiques aux valeurs des paramètres ou à l'ordre des opérations près. TOBIAS a été conçu pour aider l'utilisateur dans sa démarche de conception de tests similaires. L'utilisateur doit fournir une description abstraite des séquences de tests (*schéma de test*). Chaque schéma est « déplié » par TOBIAS et transformé en tests exécutables.

TOBIAS prend en entrée un diagramme de classes et en extrait les classes et signatures des méthodes. L'utilisateur sélectionne des ensembles de valeurs pertinentes pour chaque paramètre de chaque opération/méthode. Il peut aussi identifier et regrouper une ou plusieurs méthodes sous une même étiquette (appelée « groupe »). A partir de ces éléments, des schémas de test peuvent être définis. Un schéma est une forme d'expression régulière sur les opérations et les groupes.

Soit **S1** un schéma visant à tester l'ajout de nouveaux éléments dans le système après initialisation.
S1 : « `Initialiser(); AjouterGr` » avec « `AjouterGr =` {`Ajouter(x)|x`∈{`1,2,3,4,5`}} ».

`AjouterGr` est un groupe composé de 5 instanciations différentes de la méthode `Ajouter`. A partir des valeurs des paramètres données ci-dessus et de **S1**, TOBIAS génère 5 séquences :

```
Initialiser(); Ajouter(1)
Initialiser(); Ajouter(2)
...
Initialiser(); Ajouter(5)
```

Les groupes permettent aussi de rassembler des méthodes différentes sous une même étiquette. Soit **S2** un schéma pour tester l'évolution du système selon les opérations `Ajouter` et `Enlever`.
S2 : « `Initialiser(); Modifier^1..2` » avec

`Modifier = {Ajouter(x) | x ∈{1,2,3,4,5}} ∪ {Enlever(y) | y ∈{1,3,5}}`

Les séquences de **S2** initialisent le système puis font un ou deux appels de méthodes du groupe `Modifier` (8+8*8=72 séquences).

```
Initialiser(); Ajouter(1)
...
Initialiser(); Enlever(5)
Initialiser(); Ajouter(1); Ajouter(1)
...
Initialiser(); Enlever(5); Enlever(5)
```

3-2. Les faiblesses de TOBIAS

TOBIAS permet de décharger le testeur des tâches répétitives et dégage donc plus de temps pour les aspects créatifs de la génération de tests. L'expérience décrite dans [12] a comparé la génération de 4 320 séquences de test avec TOBIAS à la production manuelle de 45 tests. Elle a montré que la conception des deux suites de test nécessitait un effort comparable, qu'elles offraient le même taux de couverture, mais que les tests TOBIAS permettaient de trouver plus d'erreurs que l'autre.
Toutefois, deux évolutions de TOBIAS sont souhaitables : une aide à la sélection des valeurs, et l'élimination de tests non conformes.

La sélection des valeurs

Le principe de TOBIAS est d'exploiter l'expertise du testeur pour la sélection de valeurs. C'est un processus manuel. Il est intéressant de proposer des moyens automatiques de sélection des données.

Elimination de tests non conformes

Parmi l'ensemble des séquences obtenues à partir d'un schéma, toutes ne sont pas forcément pertinentes et correctes. En effet, la génération est systématique et ne prend en compte que la signature des méthodes et les valeurs choisies par l'utilisateur. Des éléments plus précis de la spécification ne sont exploités. Ainsi, on peut obtenir des tests non conformes vis à vis de diagrammes d'états-transitions ou de pré-condition.

Soit **S3** : « `Initialiser(); Ajouter; Modifier^1..3` ». **S3** permet de produire 5*(8+8*8+8*8*8)= 2 920 séquences d'appels. Parmi celles-ci, 702 violent les pré-conditions des opérations. Ces séquences sont inutiles puisqu'elles ne permettent pas de conclure sur la conformité du système. Par exemple, les séquences ci-dessous violent les pré-conditions de `Enlever` car on tente de retirer plus d'éléments que ceux présents dans les tampons.

```
Initialiser(); Ajouter(1); Enlever(5)
Initialiser(); Ajouter(2); Enlever(3); Ajouter(2); Ajouter(4)
```

4. UCASTING ET TOBIAS

UCASTING est un outil de synthèse de test développé par l'IRISA et AQL [15]. Basé sur l'outil CASTING, UCASTING exploite des spécifications UML complétées par des contraintes exprimées à l'aide d'un sous-ensemble d'OCL.

4-1. UCASTING

CASTING s'inspire des travaux de Dick et Faivre [3]. A partir de la spécification formelle de l'application à tester et d'une stratégie de test définie par l'utilisateur, CASTING construit un graphe de test. Il cherche ensuite les valeurs des paramètres permettant d'atteindre chaque transition de ce graphe. UCASTING quant à lui exploite les diagrammes d'états-transitions (Fig. 2) ainsi qu'une spécification en termes d'invariant, de pré et post-conditions.

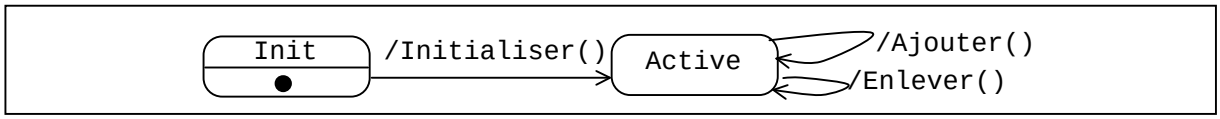

Fig. 2 Comportement du système de tampons

UCASTING construit la séquence de transitions permettant d'atteindre chaque transition. Par exemple, pour `/Enlever()`, UCASTING construit la séquence `/Initialiser();/Ajouter();/Enlever()`. L'outil déduit la contrainte correspondant à l'enchaînement des pré et post-conditions, puis la résout pour choisir des valeurs de test permettant d'exécuter les transitions (ici, « `Initialiser(); Ajouter(5) ; Enlever(5)` »).

UCASTING permet aussi l'animation de la spécification. Le solveur de contraintes permet de sélectionner des valeurs pour compléter des contraintes partiellement instanciées ou de décider de la conformité d'une contrainte complètement instanciées.

4-2. Principe de la connexion TOBIAS-UCASTING

Nous avons étudié trois moyens de combiner TOBIAS avec UCASTING.

Valider une séquence de test

UCASTING permet de vérifier la cohérence d'une séquence de test avec la spécification. Il est possible :

1. de s'assurer qu'aucune opération n'est appelée en dehors de sa pré-condition,

2. de garantir que des séquences sont bien conformes au diagramme d'états-transitions spécifié par l'utilisateur.

On peut filtrer les séquences déduites d'un schéma, pour ne conserver que celles satisfaisant la spécification.

Générer des données de tests pour une séquence donnée

Pour être transformé en tests exécutables, un schéma de test doit comprendre toutes les instanciations nécessaires. Une idée est de produire des schémas sans valeurs, puis de fournir des séquences d'appels partiellement instanciées à UCASTING. Ce dernier décide alors de leur validité et le cas échéant produit des valeurs manquantes.

Soit le schéma **S4** : « `Initialiser()^0..1; Modifier^2..2` ». Le dépliage de **S4** sans les valeurs des paramètres permet d'obtenir les 8 séquences suivantes au lieu des 2*(8*8) = 128 attendues :

1. `Ajouter(); Ajouter()`
2. `Ajouter(); Enlever()`
3. `Enlever(); Ajouter()`
4. `Enlever(); Enlever()`
5. `Initialiser(); Ajouter(); Ajouter()`
6. `Initialiser(); Ajouter(); Enlever()`
7. `Initialiser(); Enlever(); Ajouter()`
8. `Initialiser(); Enlever(); Enlever()`

UCASTING élimine les séquences *1* à *4* car elles ne sont pas conformes au diagramme d'états-transitions, et les séquences *7* et *8* car `Enlever` figure avant `Ajouter` (pas d'instanciation possible). Pour les séquences *5* et *6*, UCASTING propose des valeurs de paramètres.

Cette utilisation offre deux avantages. UCASTING élimine au plus tôt des séquences non conformes (les 6 séquences non conformes correspondent à 88 séquences une fois dépliées). D'autre part, UCASTING calcule automatiquement des valeurs satisfaisant les séquences valides. L'utilisateur bénéficie ainsi d'un moyen automatique de sélection de valeurs de test.

Vers une génération des données de test guidée

UCASTING peut aider l'utilisateur à choisir les valeurs des paramètres pour des schémas partiellement instanciés. Il fournit alors une seule instanciation. Il est possible de le contraindre dans le choix des valeurs. Soit **S5** : « `Initialiser();Ajouter(x);Enlever(y)` ». Si UCASTING a généré les valeurs 5 pour `x` et 1 pour `y`, il est intéressant de produire une nouvelle séquence en ajoutant la contrainte : « $x \neq 5 \wedge y \neq 1$ ». Ceci force UCASTING à générer de nouvelles valeurs.

5. EVALUATION DE LA CONNEXION

La collaboration entre TOBIAS et UCASTING a été mise en œuvre. Sur demande de l'utilisateur, TOBIAS communique les informations à UCASTING, récupère et traite le résultat en éliminant et/ou complétant les séquences. Pour que la collaboration soit effective, l'utilisateur doit fournir une spécification formelle comprenant :

- les pré et post-conditions des opérations,
- l'invariant d'état de la classe traitée, et
- un diagramme d'états-transitions décrivant le comportement.

Malheureusement, UCASTING gère mal les spécifications non-déterministes (c'est le cas de la spécification donnée Fig. 1). Une façon de procéder est donc de « déterminiser » la spécification, ce qui demande un travail supplémentaire et réduit le nombre d'implantations possibles. Une autre façon est « d'externaliser » les résultats comme paramètres des méthodes.

Nous avons construit une implantation de la spécification donnée Fig. 1, puis nous avons évalué les trois connexions présentées. L'exécution d'UCASTING et de TOBIAS s'est faite sur un PIII 700Mhz sous Windows 2000 et l'exécution des tests sous VDMTools se fait sur un Sun Fire 280R.

TOBIAS sans UCASTING

Le schéma **S3** permet d'obtenir 2 920 séquences de test en quelques secondes. Elles ont été exécutées dans l'atelier VDM en 7 minutes.

Cet atelier permet d'évaluer une certaine couverture des tests. La couverture d'une opération est le nombre d'expressions (présentes dans le code et dans les pré et post-conditions) qui sont évaluées par les tests divisé par le nombre total d'expressions qui définissent l'opération. Les tests exécutés couvrent 100% de la spécification.

L'analyse des résultats est rendu difficile par les messages d'erreurs associés aux 702 qui violent des pré-conditions (et ne permettent pas de conclure). Il faut étudier chaque message d'erreur renvoyé par l'atelier VDM pour déterminer s'il s'agit d'une erreur de l'implantation ou d'un test non conforme.
Toutefois, l'analyse a mis en évidence une erreur dans le code de l'opération `Enlever`. Dans notre implantation, sous certaines conditions, il était possible de rendre la taille d'un tampon négative avec le schéma « `Initialiser(); Ajouter(2); Ajouter(3);Enlever(4)` » (ce qui n'est pas conforme à la spécification).

Utilisation de UCASTING pour la génération de données
Déplier **S3** sans les paramètres permet d'obtenir 14 séquences différentes. L'instanciation de ces séquences par UCASTING n'a pris qu'une dizaine de secondes. Les 14 tests ont été exécutés sous VDMTools en une seconde.
Malheureusement, UCASTING a toujours instancié l'opération `Ajouter` avec la valeur 5. Si cette valeur est intéressante, elle ne permet pas de détecter l'erreur décrite au paragraphe précédent (bien que les séquences aient couvert 100% de la spécification).

Utilisation de UCASTING pour l'élimination de séquences non conformes
Pour éliminer les séquences non conformes à la spécification, UCASTING a été utilisé pour filtrer les 2 920 séquences de **S3**. L'étape de filtrage a pris 39 minutes et a éliminé les 702 séquences identifiées comme invalides. L'exécution des tests sous VDMTools a pris environ 1 minute soit 6 fois plus vite que l'exécution des 2 920 tests. L'analyse des résultats est facilitée par le fait qu'il n'y a pas de verdicts liés à la non-conformité des séquences. Si le temps d'exécution des tests est meilleur, cette approche demande plus d'efforts de l'utilisateur que la précédente (il faut décrire toutes les valeurs souhaitées) et requiert un temps de calcul global beaucoup plus important.

Utilisation de UCASTING pour la génération de tests à l'aide de séquences partielles
Soit **S4** : « `Initialiser(); AjouterPartielGr; ModifierPartielGr^1..3` » avec « `AjouterPartielGr = {Ajouter(n)| n="nb"}` » et `ModifierPartielGr` un groupe précédemment défini.

TOBIAS génère 258 séquences de test partiellement instanciées à partir de **S4**. Ces séquences ont été envoyées à UCASTING pour qu'il les filtre et les complète. Cette étape n'a pris que 3 minutes et 145 séquences ont été éliminées. Les séquences produites ont été exécutées en 2 secondes sous VDMTools. Elles couvrent 100% de la spécification et détecte l'erreur ci-dessus. L'analyse des résultats a été facilitée par la taille raisonnable de la suite de tests.

Cette approche semble offrir le meilleur compromis entre l'effort à fournir, le temps de calcul et les résultats obtenus. Elle permet à l'utilisateur de se concentrer sur les points qui lui semblent importants. Il n'a qu'à spécifier les valeurs des opérations qu'il souhaite tester et laisser le soin à l'outil de compléter les valeurs des paramètres des autres opérations.

6. CONCLUSION

Cette étude a montré comment TOBIAS pouvait se combiner avec UCASTING, pour assister le testeur dans la sélection des données de test. A terme, il s'agit de lui proposer des outils qui ne substituent pas à lui, mais lui permettent d'exploiter son expérience, tout en le déchargeant de tâches moins créatives.

Nous avons présenté, implanté et évalué trois approches pour combiner ces deux outils :
- UCASTING peut filtrer les séquences de test complètement instanciées de TOBIAS, en vérifiant si elles sont conformes au diagramme d'états-transitions, à l'invariant et aux pré et post-conditions.
- UCASTING peut générer les valeurs des paramètres à partir d'un dépliage partiel d'un schéma de test.
- UCASTING peut compléter des séquences de tests partiellement instanciées en fonction des valeurs des autres paramètres, et éliminer les séquences impossibles.

La plus prometteuse des trois approches est la dernière. Elle permet de maîtriser une partie de l'explosion combinatoire due aux combinaisons des valeurs des paramètres, et le temps de filtrage et complétion des séquences par UCASTING est moins important qu'avec la première approche. L'intérêt est double puisque cette technique permet à l'utilisateur de ne se concentrer que sur la ou les méthodes qu'il souhaite tester tout en évitant des temps de calcul trop importants et une analyse des résultats trop fastidieuse dus à un grand nombre de séquences de test.

Limites de UCASTING. Au cours de cette expérimentation, nous avons été confrontés à deux limitations de la version actuelle d'UCASTING :

- Le non-déterminisme interne des opérations n'est pas supporté. Une stratégie consiste à externaliser certaines variables en paramètre, permettant à UCASTING de les instancier.
- L'outil ne travaille que sur des variables entières et ne supporte pas de structures de données. Les prochaines versions d'UCASTING devraient étendre le langage de spécification.

Autres approches. Il existe d'autres techniques de générations de tests à partir de spécifications formelles en termes de pré et post-conditions. L'outil BZ-TT [10], basé sur les travaux de Dick et Faivre [3], permet de générer plus de cas de test par une exploration systématique des cas aux limites. Il pourrait donc être intéressant de voir quels pourraient être les apports d'une utilisation conjointe de TOBIAS et de BZ-TT.

L'outil TestEra [11], basé sur le langage Alloy et son analyseur de contraintes [7], présente des similarités avec TOBIAS. Il utilise une spécification orientée modèle comme oracle du test, et génère des données de test à partir d'une spécification des paramètres d'entrée des méthodes. Ceci permet l'utilisation en entrée de types structurés (listes, arbres, ...). L'utilisation de types structurés constitue un progrès pour TOBIAS, même si dans TestEra, elle se fait aux dépens de la facilité d'expression des données de test.

REFERENCES

[1] B. Beizer : Software Testing Techniques. Van Nostrand Reinhold, 1990.
[2] P. Bontron, O. Maury, L. du Bousquet, Y. Ledru, C. Oriat, et M.-L. Potet : TOBIAS : un environnement pour la création d'objectifs de tests à partir de schémas de tests. In ICSSEA, déc. 2001.
[3] J. Dick and A. Faivre : Automating the generation and sequencing of test cases from model-based specifications. In FME'93: Industrial-Strength Formal Methods. LNCS 760, Springer, April 1993.
[4] L. du Bousquet, H. Martin, and J.-M. Jézéquel. Conformance Testing from UML specifications, Experience Report. In Gesellschaft für Informatik (GI), p-UML workshop, Lecture Notes in Informatics. vol. P-7, 2001.
[5] J. Fitzgerald and P. G. Larsen : Modelling Systems -Practical Tools and Techniques in Software Development. Cambridge Univ. Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.
[6] The VDM Tool Group : VDM-SL Toolbox User Manual. Technical report, IFAD, Oct. 2000. ftp://ftp.ifad.dk/pub/vdmtools/doc/userman_letter.pdf.
[7] D. Jackson, I. Shlyakhter, and M. Sridharan : A micromodularity mechanism. In Proc. 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering, Vienna, Australia, sep. 2001.
[8] T. Jéron and P. Morel : Test Generation Derived from Model-checking. In Computer Aided Verification (CAV). LNCS 1633, Springer, 1999.
[9] C. B. Jones : Systematic Software Development Using VDM (2nd Edition). Prentice-Hall, London, 1990.
[10] B. Legeard, F. Peureux, and M. Utting : Automated boundary testing from Z and B. In Formal Methods Europe (FME), 2002, Springer.
[11] D. Marinov and S. Khurshid : TestEra: A novel framework for automated testing of Java programs. In Proc. 16th IEEE Int. Conf. on Automated Software Engineering (ASE), Nov. 2001.
[12] O. Maury and Y. Ledru : Using TOBIAS for the automatic generation of VDM test cases. In VDM workshop, Danemark, 2002.
[13] J. Tretmans and A. Belinfante : Automatic Testing with Formal Methods. In 7th European Int. Conf. on Software Testing, Analysis & Review, EuroStar Conferences, Galway, Ireland, 1999.
[14] L. Van Aertryck, M. Benveniste, and D. Le Métayer. Casting: A formally based software test generation method. In The 1st Int. Conf. on Formal Engineering Methods (ICFEM), IEEE, Hiroshima, 1997.
[15] L. Van Aertyck, Th. Jensen. UML-CASTING: Test synthesis from UML models. Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL), Janv. 2003