Génération automatique de tests pour les systèmes du RGT

Sandro Mazziotta
Institut Eurécom,
2229 route des crêtes, B.P. 193,
06904 Sophia Antipolis CEDEX, France.
Tel: 93.00.26.61, email: mazziott@eurecom.fr

Résumé

Les spécifications formelles ne sont pas une fin en soi. Il faut plutôt les considérer comme l'étape initiale nécessaire pour effectuer sur la spécification d'un système et ses implémentations futures des activités telles-que la validation, la vérification, le test et déboguage, ... Aprés avoir présenté le formalisme des comportements d'objets gérés défini dans le cadre de TIMS [1], cet article propose des ébauches de solutions pour la génération de suite de tests de spécifications de comportements d'objets gérés du RGT.

1 Introduction

L'objectif principal du projet TIMS est la compréhension des modèles d'information du RGT et la proposition de solutions aux problèmes qui leurs sont liés. Le problème essentiel réside dans le fait que le comportement des objets gérés est défini en prose dans les spécifications GDMO[2, 11]. Ceci interdit l'automatisation de toutes étapes ultérieures telles-que: la validation, la vérification, le test et déboguage, ... C'est pourquoi une grande part des activités de recherche dans ce domaine réside en la proposition de nouveaux formalismes ou en l'adaptation de formalismes existants permettant la description formelle de ces comportements. Pour une bibliographie complète, le lecteur est prié de se référer à [6].

Cependant la plupart des approches déjà existantes ne convenait pas à un des objectifs principaux du projet: la simulation des modèles d'informations du RGT. C'est pourquoi il a été défini dans le cadre du projet, d'une part un formalisme pour la description des comportements des objets gérés et d'autre part, une sémantique opérationnelle [15]. L'étape de formalisation apporte beaucoup en ce qui concerne la compréhension des modèles d'information et le fait de pouvoir simuler ces modèles est déjà un gain intéressant. Il convient

1. TIMS signifie TMN-based Information Model Simulator (Simulation de modèle d'informations basé sur le RGT). Ce projet est une collaboration entre l'Institut Eurécom et Swiss Telecom PTT. Il est financé par le projet F&E-288 de Swiss Telecom PTT.

maintenant d'appliquer les résultats obtenus pour automatiser des tâches qui jusqu'à présent ne sont encore effectuées que manuellement.

Parmi elles, le test est une étape cruciale dans le cycle de vie d'une application (d'un point de vue économique). Le but du test est de trouver des erreurs dans l'implémentation. Il est généralement impossible de prouver qu'une implémentation est correcte. L'un des problèmes majeurs est alors de décrire une suite de tests suffisamment complète pour avoir une certaine confiance dans l'implémentation.

L'objectif du papier est de proposer des ébauches de solutions pour la génération de suite de tests de spécifications de comportements d'objets gérés du RGT.

Plan du Papier Le papier est organisé comme suit:

- La section 2 présente brièvement le formalisme des comportements d'objets gérés tels qu'ils sont définis dans le cadre du projet.
- La section 3 présente les différentes techniques de génération de tests et celles que l'on retient pour les comportements de la section précédente.
- La section 4 présente un premier type de test: le test de couverture des comportements.
- La section 5 présente un deuxième type de test: le test de robustesse.
- La section 6 conclut ce papier et propose les extensions qui sont envisagées à ce travail.

2 Les comportements d'objets gérés

Un comportement d'objet géré est défini comme étant une interaction (échange de messages) entre des objets à la réception d'un message donné (le trigger) sur une des interfaces de l'objet. Cet échange de message a lieu si au préalable la garde (condition booléene qui rend possible l'exécution d'un comportement) est évaluée à vrai.
A noter que l'exécution d'un bout de code à la réception d'un événement est classiquement utilisé dans les systèmes informatiques. Par exemple on retrouve aussi ce mécanisme dans les bases de données actives pour les règles ECA (Event–Condition–Action) [8]. Dans notre cas, on utilise ce mécanisme dans un but de rendre la spécification exécutable [7] pour faire de la simulation alors que dans le cas des bases de données, il est utilisé pour faire de la gestion et/ou du contrôle.

De plus, pour chaque comportement, on donne la possibilité de spécifier des assertions (pré et post-conditions qui vérifient que le système est dans un état correct avant et après l'exécution d'un comportement) et ce à fin de fixer des propriétés au niveau de chaque comportement qui seront vérifiées durant l'exécution de la simulation.

En résumé, un comportement est défini par un quintuplet
$(Label, Guard, Pre, Body, Post)$ avec

- $Label$: identificateur de comportement,
- $Guard, Pre, Post$: expression booléene,
- $Body$: Le corps d'un comportement est un bout de code *Scheme* [3] pour lequel il n'y a aucune condition particulière [2].

2. Pour spécifier des tests et des effets, une API de manipulation des objets est disponible

Comme le souligne [14], l'approche qui a été suivie est dite orientée modèle (le système est décrit en terme de données et d'opérations) par opposition à l'approche orientée processus comme notamment les FDTs (Techniques de Description Formelle) standardisées (LDS, LOTOS) ou bien CCS.
Toutes deux décrivent une machine abstraite en terme d'états et de transitions possibles entre ces états. La différence essentielle entre les deux approches est dans la présentation de cette machine abstraite. Alors que pour l'approche orientée modèle, on définit pour chaque action quels changements d'états peuvent survenir, dans l'approche orientée processus, on donne pour chaque état les transitions qui sont possibles.

3 Génération automatique de suites de tests

On dénombre peu de travaux dans le domaine du test et de l'administration de réseaux OSI. Les plus connus [5, 13, 10] ne parlent pas ou peu des aspects qui font l'objet de ce papier, le test de comportement. En effet dans ces approches, l'effort a plutôt été porté sur le modèle d'information statique (définitions statiques des spécifications GDMO/ASN.1). On suppose, dans le contexte de ce papier, que le modèle d'information statique ainsi que le protocole (CMIP dans le cas du RGT) sont corrects. C'est à dire qu'ils n'introduisent pas d'erreurs. De cette manière, les fautes obtenues ne sont dues qu'à des fautes de comportements (des fautes qui ne se réfèrent qu'aux aspects dynamiques du système).

Les techniques classiques pour la génération de suites de tests peuvent aisément se diviser en deux selon que l'on dispose ou pas de la connaissance interne de l'implémentation:

- Le test fonctionnel (test de boîte noire) fait abstraction de la structure interne de l'implémentation et se concentre sur les interactions entre l'environnement et les interfaces observables de l'implémentation.
- Le test structurel (test de boîte blanche) quant à lui, au contraire, se sert du code de l'implémentation pour la dérivation des suites de tests.

Dans le domaine des télécommunications et particulièrement pour les protocoles, c'est le test de boîte noire qui est utilisé. Le plus souvent les spécifications sont écrites (ou disponibles) dans une FDT standardisée. On représente ensuite, du point de vue du comportement, ces spécifications, leurs implémentations, les objectifs de tests et les tests eux mêmes à l'aide de modèles tels-que: les automates à entrées sorties ou les systèmes de transitions étiquetées [12].

Dans le domaine du test de logiciel, le test structurel est le plus utilisé. Les techniques de générations se basent soit sur le flux de contrôle d'exécution soit sur le flux de données. On génère par exemple des tests pour chaque instructions, branches (dans une expression conditionnelle) ou chemins d'un programme.

Dans notre cas, l'objectif est de dériver des tests à partir des spécifications de comportements tels qu'ils sont définis dans la section 2. L'idée est de combiner les techniques du test de protocole et celles du test de logiciel. Les premières car le modèle que l'on utilise et celui des FDTs standardisées sont proches. La différence essentielle est que dans le test de protocole, la spécification est considérée correcte et c'est des implémentations qui sont testées alors que dans le contexte de ce papier, au contraire, on essaie de trouver les erreurs

dans la spécification. Les secondes car les spécifications sont exécutables (contiennent du code).

A noter que du fait que l'on se trouve dans un contexte de télécommunications: le RGT. On va s'appliquer à adopter la méthodologie du test de protocole (issues de [1]) et ce à fin de pouvoir réutiliser les tests générés (par exemple pour tester des implémentations futures). Par exemple, uniquement les interfaces standardisées sont considérées pour les interactions avec l'environnement.

4 Les tests de couverture des comportements

Ce type de test consiste à exécuter chacun des comportements et ce à fin de vérifier que chaque comportement est déclenchable. Dans notre cas, il s'agit de vérifier que le concepteur de la spécification n'a pas commis d'erreurs dans l'expression de ces gardes (clause guard cf section 2). L'intérêt de ces tests réside aussi dans le fait que couplé (ou injecté) dans le simulateur, ils vont permettre de valider les comportements dans leur ensemble (l'exécution de la clause body encapsulée par les vérifications assertionnelles des clauses pré et post).

De manière pratique, il s'agit de calculer pour chaque comportement et plus particulièrement pour chacune des gardes, l'ensemble des messages à envoyer au système, pour le configurer de manière à ce que le comportement puisse s'exécuter (et donc que la garde soit évaluée à vrai). A partir de l'état initial $MIB^{initial}$, il s'agit de trouver la séquence de transitions qui va mettre le système dans l'état MIB^{guard}. Cet état est une fonction f de l'état initial et de la garde du comportement que l'on cherche à exécuter. Avant de penser à implémenter cette fonction, il faut résoudre un certain nombre de problèmes:

1. Quel est l'état $MIB^{initial}$?
   A priori, deux solutions distinctes sont acceptables : soit la MIB est vide, soit la MIB est déjà configurée et elle est dans un état stable (MIB^{config}). En fait si on analyse la spécification, on remarque assez aisément qu'il existe plusieurs états MIB^{config} : MIB_1^{config}, MIB_2^{config},... qui correspondent à des configurations précises pour lesquelles le concepteur de la spécification définit des comportements.
   Dans le premier cas, il s'agit d'être capable de déduire l'ensemble des transitions pour mener le système de l'état MIB^{vide} à l'état MIB^{config}. Dans ce cas, il n'y a pas de contraintes d'utilisation mais le problème devient difficile à résoudre. Il faudra par exemple déduire des post-conditions de la spécification comment configurer la MIB.
   Dans le second cas, la configuration du système est à la charge de l'utilisateur. Pour tester un comportement donné, il faudra qu'il spécifie l'état MIB_i^{config}. C'est une contrainte forte mais on simplifie le problème.
2. Le résultat de la fonction f nous donne un ensemble de transitions. Ce qui dans l'absolu nous donne un grand nombre de chemins (exécutions) possibles. Cependant on peut d'ores et déjà dire que tous ces chemins ne sont pas solutions à notre problème. Il peut y avoir par exemple une relation d'ordre entre les différentes transitions (par exemple la configuration d'un objet ne peut avoir lieu avant sa création). Ainsi il est important de vérifier s'il existe de telles propriétés dans le résultat de f. A noter que compte tenu des objectifs de ce type de test, il n'est pas nécessaire de générer toutes les solutions; il suffit d'en trouver une.

5 Les tests de robustesse

En complément des tests précédents, il est prévu de générer des tests avec des objectifs plus ambitieux. On se propose, par exemple, de tester les spécifications par des tests de robustesse. Plus précisément, on se propose de tester la robustesse par rapport à deux propriétés: le nondéterminisme et la tolérance aux fautes.

- Pour le cas du nondéterminisme, on choisit pour stratégie de maximaliser la manifestation du nondéterminisme. Ce qui aura pour effet, suivant notre modèle, de provoquer de l'entrelacement entre les comportements (exécution concurrente de plusieurs comportements). L'idée sous-jacente, est de vérifier si l'exécution ne va pas entraîner par exemple une levée d'assertions (invalidation de post-condition). Ainsi on vérifie si le concepteur de la spécification a bien protéger son système.
- Pour le cas de la tolérance aux fautes, on choisit pour stratégie l'injection de fautes. Pour la génération de ces fautes, on envisage de s'inspirer du "test par mutation" [4]. Ce type de test consiste à dériver des spécifications "mutantes" à partir de la spécification originale. Chaque mutant diffère de l'originale par une et une seule modification. L'idée principale réside dans le fait que chacun des mutants correspond à une erreur possible (classique) de spécification. Les tests sont générés pour distinguer les mutants de la spécification originale dans l'espoir qu'ainsi la spécification aura suffisamment été éprouvée.

Dans les deux cas, c'est l'étude des assertions (pré et post-condition) qui va diriger la génération du test et notamment, il s'agit de trouver des contextes qui rendent l'assertion fausse.

6 Conclusion

Ce papier présente dans un premier temps le modèle de comportements des objets gérés. Dans un second temps, il énumère les approches déjà existantes pour la génération de suites de tests et le modèle que l'on a retenu pour tester les comportements. Il présente enfin des ébauches de traitement pour deux types de tests: le test de couverture des comportements et le test de robustesse.

Afin d'optimiser la correction de fautes, l'écriture d'un debugger [14] et son couplage avec le module de test est considéré. La compilation des séquences de tests en un langage de description de tests standardisé comme TTCN[9] pourrait aussi être considérée. L'objectif est double: les tests pourraient ainsi être appliqués à des implémentations réelles mais aussi ils pourraient être diffusés (par l'intermédiaire d'organismes de standardisation).

Références

[1] Information Technology - Open Systems Interconnection - Conformance Testing, Methodologie and Framework, ISO/IEC IS 9646.

[2] Clemm (Alexander) et Festor (Olivier). – Behavior, Documentation, and Knowledge : An Approach of the Treatement of OSI–Behavior. *In : 4th International Workshop on Distributed System Operations and Management*.

[3] Clinger (W.) et Rees (J.). – $Revised^4$ Report on the Algorithmic Language Scheme. *ACM Lisp Pointers*, vol. 4, n° 3, 1991. – Available at http://www.cs.indiana.edu/scheme-repository/doc/standards/r4rs.ps.gz.

[4] DeMillo (R. A.) et Offutt (A. J.). – Experimental results of automatically generated adequate test sets. *In : Proceedings of the Sixth Annual Pacific Northwest Software Quality Conference*. pp. 209–232. – Portland OR, September 1988.

[5] Methodology for Testing Conformance to Managed Objects, EWOS PT N 028-Draft 9 bis , 1995.

[6] Festor (Olivier). – *Formalisation du Comportement des Objets Gérés dans le Cadre du modèle OSI*. – Thèse de PhD, Université Henri Poincaré, Nancy I, Centre de Recherche en Informatique de Nancy (CRIN), Octobre 1994.

[7] Fuchs (Norbert E.). – *Specifications are (preferably) executable*. – Rapport technique n° 92, University of Zurich (CS Dept.), 1992. Available at ftp://ftp.ifi.unizh.ch/pub/techreports/.

[8] Geppert (Andreas), Gatziu (Stella), Dittrich (Klaus R.), Fritschi (Hans) et Vaduva (Anca). – *Architecture and Implementation of the Active Object-Oriented Database Management System SAMOS*. – Rapport technique, University of Zurich, Dept. of Computer Science, 1995. Available at http://www.ifi.unizh.ch/techreports.

[9] ITU-T. – ITU-T Recommendation X.292-92: OSI Conformance Testing Methodology and Framework for Protocol Recommendations for CCITT Applications - The Tree and Tabular Combined Notation., 1992.

[10] Kaboré (Paul). – *Une approche de test de conformité des systèmes d'administration de réseaux.* . – Thèse de PhD, Université Henri Poincaré, Nancy I, Centre de Recherche en Informatique de Nancy (CRIN), 1995.

[11] Kilov (Haim). – Understand, Specify, Reuse: precise specification of behavior and relationships. *In : Proceedings DSOM'92*. – Munich, Germany, 1992.

[12] Phalippou (Marc). – *Relation d'implantation et hypothèses de test sur des automates à entrées et sorties*. – Thèse de PhD, Université de Bordeaux, 1994.

[13] Testing Managed Objects - An Object Oriented Formal Approach, PROST Objects consortium, Logica BT NCC , 1995.

[14] Sidou (Dominique). – A Generic and Executable Model for the Specification and Validation of Distributed Behaviors. *In : TreDS'96 : Trends in Distributed Systems Workshop, published in LNCS 1161*. – available at http://www.eurecom.fr/~tims/papers/treds96.ps.gz.

[15] Sidou (Dominique), Mazziotta (Sandro) et Eberhardt (Rolf). – TIMS : a TMN-based Information Model Simulator, Principles and Application to a Simple Case Study. *In : Sixth International Workshop on Distributed Systems : Operations & Management*. IFIP / IEEE. – Ottawa - Canada, 1995. Available at http://www.eurecom.fr/~tims/papers/dsom95-paper.ps.gz.