Sélection des tests à partir de spécifications orientées objets

par El Houssain Htite, Rachida Dssouli et Gregor v. Bochmann

Publication # 871

Département d'informatique et de recherche opérationnelle

Université de Montréal

Juillet 1993

Sélection des tests à partir de spécifications orientées objets

El Houssain Htite, Rachida Dssouli et Gregor v. Bochmann

Université de Montréal
Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences, C.P. 6128, Succ "A"
Montréal, (Québec) Canada H3C 3J7

Email: {dssouli} @iro.umontreal.ca
fax: (514) 343-5834

Résumé

La sélection des séquences de test pour la conformité des protocoles de communications est devenue un champ où la recherche est très active. Plusieurs méthodes ont été développées afin de permettre la sélection et la génération des séquences de test pour des systèmes logiciels et matériels. En effet, ses méthodes ont la caractéristique de tester des systèmes modélisés par une seule machine à états finis (FSM "Finite State Machine"). Avec l'approche orientée objet, la spécification d'un système peut être vue comme un ensemble dynamique de machines à états finis. Dans l'objectif de pouvoir tester de tels systèmes, d'autres méthodes et stratégies doivent être développées. Dans cet article, nous nous intéressons à la sélection des tests à partir de spécifications orientées objets. Pour se faire, nous allons définir un modèle de fautes basé sur les techniques de description formelles orientées objets comme MONDEL [Boch 90, Boch 91]. Ensuite, on va étudier les différents aspects qui influencent le test orienté objet et enfin, nous proposerons une méthodologie de sélection de test destinée aux système orientés objets.

1. Introduction

La vérification de la conformité d'une implantation par rapport à sa spécification de référence est d'une importance primordiale. Le test est l'un des moyens pour rendre possible cette vérification. Il consiste à sélectionner des tests à partir de la spécification et les exécuter ensuite sur l'implantation afin de déceler tout comportement anormal. L'utilisation des techniques de description formelles pour la spécification des protocoles a permis de développer plusieurs méthodes de sélection des tests [Chow 78, Gone 70, Nait 81, Sabn 88]. La plupart de ses techniques de description formelles sont fondées sur le formalisme de machine à états finis

(FSM, "Finite State Machine") [Koh 78]. Le comportement d'un système dans ce formalisme est décrit par un ensemble finis d'états, un ensemble d'entrées, un ensemble de sorties, un état initial et un ensemble de transitions qui sont des paires (entrée, sortie) qui relient les états entre eux.

L'approche orientée objet basée sur le formalisme entité-relation [Chen 76, Mond 90, Boch 90, Boch 92] amène des nouveaux concepts comme l'héritage, les relations et la configuration dynamique. Cette approche possède l'avantage d'être naturelle, en effet, à chaque composante extraite du système réel correspond une entité logique faisant partie de la spécification. La spécification orientée objet d'un système peut être vue comme un ensemble dynamique de EFSM ("Extended Finite State Machine"). EFSM est une extension d'une FSM dans laquelle les transitions peuvent contenir des paramètres.

Plusieurs travaux ont porté sur le développement des langages de programmation orientées objets et sur les techniques de conception et de spécification orientées objets [boch 91]. Nous croyons que les nouveaux concepts véhiculés par l'approche orientée objet vont influencer le processus de sélection des tests. En effet, un système peut être vu comme une collection d'objets dont l'aspect dynamique ne peut plus être représenté par une seule FSM. Ce qui engendre par conséquent une limitation quant à l'utilisation des méthodes de sélection des tests classiques (tour de transition [Nait 81], séquences de distinction [Gone 70], séquences uniques d'entrées/sorties [Sabn 88] et W [Chow 78]).

Le reste de l'article sera organisé comme suit. Dans la section 2 nous allons introduire l'approche orientée objet. A la section 3 nous définirons un modèle de fautes pour les spécifications orientées objets. La section 4 décrit les différents aspects à considérer pendant la phase de sélection des tests. A la section 5 nous présenterons notre méthodologie de sélection des tests. Enfin, à la section 6 nous discuterons les différents résultats et les limitations ainsi que des idées pour des travaux futurs.

2. Approche orientée objet

Devant la complexité croissante des problèmes traités, il s’est avéré que l’approche de la programmation structurée, qui considère un programme ou un protocole comme un ensemble de procédures et un ensemble de données sur lequel agissent ces procédures, a atteint ses limites. En effet, si un langage permet un bon découpage d’une application en procédures, le moindre changement de la structure de données risque d'entraîner des changements profonds dans la structure et l’organisation de ces procédures [Mas 89].

L’approche orientée objet vient pour pallier à cet inconvénient en regroupant les données et les procédures qui les manipulent dans une seule entité qui est l’objet. Cette approche possède l'avantage d'être naturelle, la spécification se réduit à définir des entités qui simulent des objets réels du système. Plus précisément un objet se compose de trois parties (figure 1):

- **partie statique**: cette partie constitue l’ensemble des données (les attributs). La façon d'implanter ces données est invisible de l’extérieur de l’objet.
- **partie dynamique**: qui renferme un ensemble de procédures qui manipulent les données de la partie statique. Cette partie permet de décrire le comportement de l’objet.
- **partie interface**: c’est par l’intermédiaire de cet interface que l’extérieur communique avec l’objet en question. L’interface spécifie les interactions (les opérations) que l’objet peut avoir avec son environnement extérieur. En dehors de ces interactions aucune communication n’est possible.

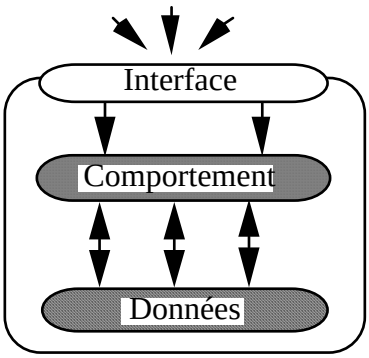


Figure 1: le concept d’objet

Cette approche permet ainsi d’assurer une abstraction des données; nul besoin de connaître l’implantation des données de l’objet pour lui demander d’effectuer une tâche (opération). Cet environnement amène donc à structurer une application en termes d’objets plutôt qu’en termes de procédures.

La programmation d’une application dans l’environnement orienté objet commence d’abord par définir les types d’objets (classes) appropriés, avec leurs opérations spécifiques. Chaque entité référencée dans le programme est un représentant (instance) d’un des types d’objet. La classe est le modèle conceptuel qui décrit l’objet. Toute création d’instance doit se faire en respectant le plan de construction du modèle de la classe à qui elle appartient.

Le mécanisme d’héritage dans l’approche orientée objet facilite le développement de nouvelles classes. Une classe (sous-classe) peut hériter les comportements d'autres classes. Une sous-classe

est donc une spécialisation de la description d'une classe (super-classe). La spécialisation peut avoir deux sens [Mas 89]:

1- **l'enrichissement**: où la sous-classe sera dotée, en plus de ce qu'elle a hérité de sa super-classe, de nouvelles variables, de nouveaux comportements.

2- **la substitution**: qui consiste à redéfinir un comportement hérité de la super-classe et qui s'avère inadéquat pour cette sous-classe.

L'héritage peut être simple ou multiple. Il est simple quand la sous-classe ne possède qu'une seule super-classe, par contre il devient multiple quand la sous-classe hérite de plusieurs super-classes. La figure 2 donne un exemple d'héritage.

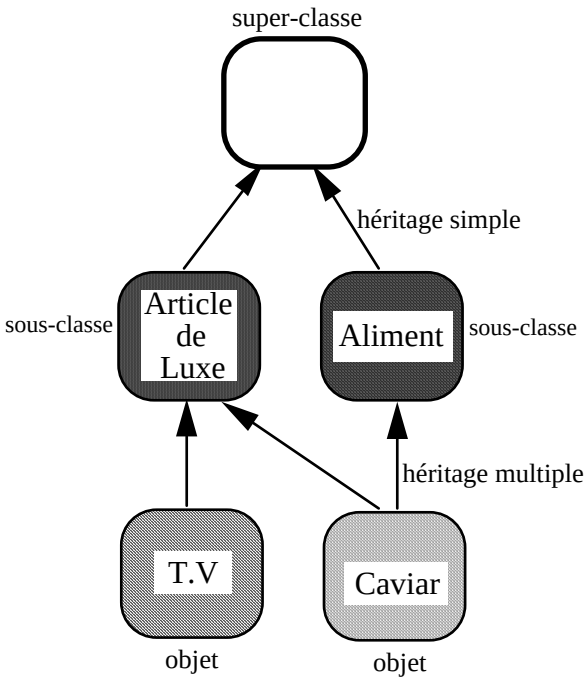


Figure 2: graphe d'héritage

Une classe peut aussi représenter des relations. Une relation est une association entre les objets. elle peut être implantée de deux façons différentes:

1- **sous forme d'attribut**: où un objet faisant partie de la relation pointe par l'intermédiaire de ses attributs les objets qui sont en relation avec lui. L'attribut doit avoir le même type que l'objet pointé.

2- **sous forme d'objet autonome**: la relation est un objet autonome qui pointe par l'intermédiaire d'attributs tous les objets concernés par cette relation. Cet objet relation n'a pas de comportement spécifique; cependant, il peut offrir des opérations permettant d'informer sur les objets qui sont concernés par cette relation.

Le choix du type de modélisation de la relation dépend en premier lieu de la nature de la relation. Il serait plus avantageux de modéliser une relation statique (relation qui relie les mêmes objets indépendamment du temps) par le type attribut, par contre si la relation est dynamique ou relie beaucoup d'objets, le type objet est alors plus intéressant. Ce choix peut aussi dépendre de la complexité de la relation (relation 1:N où un objet est en relation avec un ensemble d'objet , M:N où deux ensembles d'objets sont en relation) [Boch 92].

3. Le modèle de fautes

Il est important pendant la phase de test d'un système de pouvoir déceler la présence de toutes anomalies de ce système par rapport à sa spécification. L'ensemble des anomalies qui peuvent avoir lieu est très grand, parfois même la description de ces anomalies est très complexe. L'objectif majeur du test étant la détection de la présence de toute anomalie, or plusieurs anomalies peuvent générer la même erreur. Ce qui nous conduit à raisonner en terme de modèle de fautes qui décrit les effets des anomalies à un niveau d'abstraction plus élevé. Ainsi les séquences de test seront sélectionnées de sorte quelles détectent toutes les fautes décrites dans le modèle de fautes; ce qui a pour conséquence de réduire le nombre de possibilités à prendre en considération.

3.1. Modèle de fautes basé sur les machines à états finis

Les types de fautes à considérer pour une spécification déterministe basé sur le formalisme des machines à états finis peuvent être décrit comme suit [Boch 91a]:

a)- **faute de sortie**: qui se manifeste lorsque la sortie d'une transition de l'implantation diffère de cette même transition dans la spécification. Dans figure 3, la transition T2 dans l'implantation a une faute de sortie (e au lieu de f).

b)- **faute de transfert**: qui se présente dans le cas où l'état terminal d'une transition de l'implantation diffère de celle de la même transition dans la spécification. La transition T1 de l'implantation a une faute de transfert (l'état S0 au lieu de S1).

c)- **faute de transfert avec états supplémentaires**: certaines erreurs ne peuvent être modélisées que par des états supplémentaires avec des fautes de transferts. C'est le cas de la transition T6 qui a une faute de transfert à l'état supplémentaire S3.

d)- **faute de transitions supplémentaires ou manquantes**: dans la figure 3, la transition T7 dans l'implantation est une transition supplémentaire. La transition T9 est une transition manquante dans l'implantation.

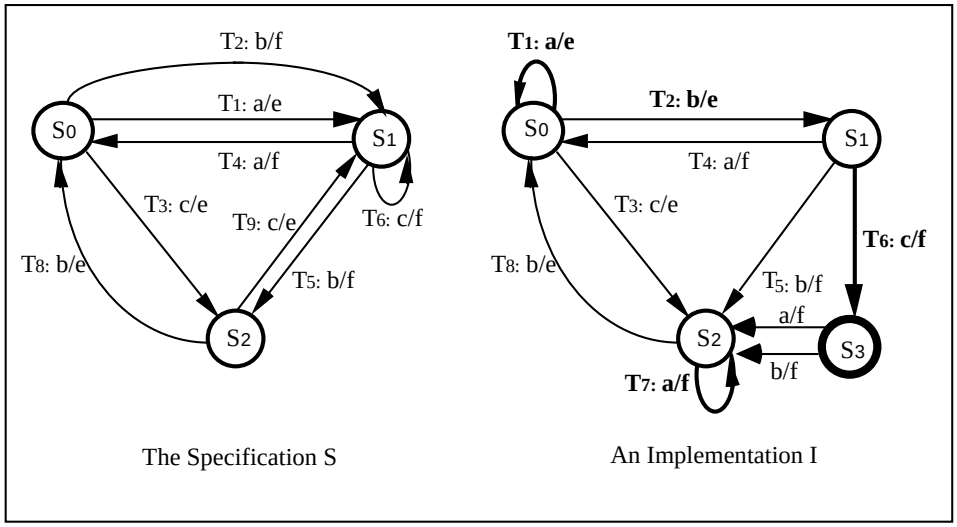


Figure 3: exemple de machines à états finis

3.2. Modèle de fautes pour les logiciels

Les types de fautes pour les logiciels peuvent être classées en deux sous-classes [Boch 91a]; les fautes de contrôle [Sari 87, Ural 91] (du point a au point c dans la classification ci-dessous) et les fautes des données (du point d au point g). Ces types de fautes sont:

a)-faute de séquencement: boucles, mauvaise expression logique, etc.

b)-faute d'arithmétique et de manipulation:

c)-faute d'appel de mauvaise fonction.

d)-faute de spécification des types de données.

e)-faute de mauvaise valeur initial.

f)-faute de référence a une mauvaise variable.

g)-faute de définition de variable sans un bon usage.

3.3. Modèle de fautes basé sur l'approche orientée objet et modèle entité-relation

Une spécification orientée objet est décrite par un ensemble d'objets et de relations entre les objets. Un modèle de fautes pour ce genre de spécification doit tenir compte des différentes caractéristiques de l'approche orientée objet. L'aspect modulaire et l'aspect configuration dynamique (au cours de l'évolution du système plusieurs objets et relations vont être créés, tandis que d'autres seront détruits) sont entre autres deux caractéristiques dominantes qu'il faut considérer. Nous définirons dans ce qui suit un modèle de fautes pour les spécifications orientées objets. Ce modèle est décrit par deux classes (fautes de configuration et fautes de comportement), où chaque classe définit un type de fautes bien déterminé.

a) Faute de configuration

Cette classe de fautes englobe toutes les erreurs dues à l'absence ou à l'existence en trop d'objets ou de relations. Tous les objets et les relations décrits par leurs types dans la spécification doivent exister dans toute implantation, l'absence ou l'existence en trop d'un objet ou d'une relation dans l'implantation peut rendre cette dernière non conforme à sa spécification. Deux sous-classes ont été identifiées:

1)- **faute d'existence d'objet**: quand un d'objet spécifié n'existe pas dans l'implantation ou quand un objet, spécifié ou non, existe en trop.

2)- **faute d'existence de relation**: quand la relation spécifiée n'existe pas dans l'implantation ou quand une relation spécifiée ou non existe en trop.

Le cas de relation qui ne relie pas les bons objets peut être vue comme une faute multiple où la première est une absence de relation et la deuxième est l'existence en trop d'une relation. La figure 3 illustre des exemples de cette classe de fautes.

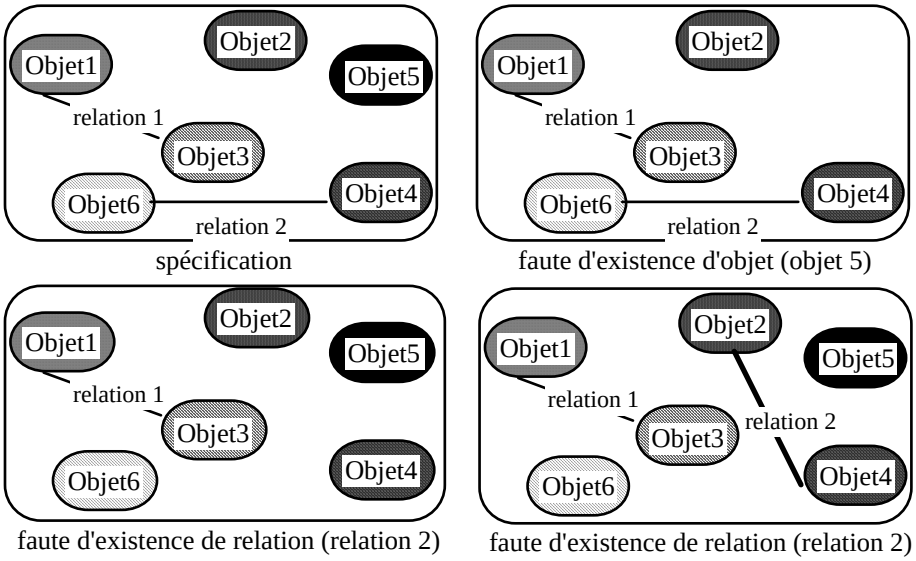


Figure 4: faute de configuration

Cette classe de fautes peut se manifester pendant l'initialisation du système, ce qui représente une faute de configuration de type statique, mais elle peut aussi avoir lieu pendant l'exécution des comportements des objets (faute de configuration de type dynamique). En effet, certains comportements d'objet peuvent introduire des changements dans la configuration du système. Ces changements font que certaines objets ou relations vont être détruits tandis que d'autres vont être créés. La faute de configuration concerne justement ces créations et ces destructions (par exemple, après un comportement désigné, l'objet A devrait être détruit alors qu'il ne l'a pas été, ou l'objet B doit être crée mais cette création n'a jamais eu lieu).

b) faute de comportement

La classe de fautes de comportement est l'ensemble des fautes propres aux comportements des objets. Nous proposons la classification suivante:

1- **mauvais appel d'opération ou d'objet**: l'interface d'un objet se compose d'un certain nombre d'opérations, au moment de son invocation, on doit préciser l'objet et l'opération voulus. Les fautes de cette sous-classe concernent les invocations (le point c dans le modèle de fautes pour les logiciels). Par exemple, l'objet A est invoqué par l'objet B pour une opération X alors qu'il devrait être invoqué pour l'opération Y (l'objet A offre les deux opérations X et Y). Une autre faute possible quand l'objet A invoque l'objet B pour l'opération Z alors qu'il devrait invoquer l'objet C (les deux objets B et C offrent l'opération Z).

2- **mauvais paramètres de l'opération**: cette sous-classe englobe les fautes relatives aux valeurs des paramètres dans les opérations (le bon objet est invoqué avec la bonne opération mais les paramètres de cette opération sont erronés). (les points d et e dans le modèle de fautes pour les logiciels).

3- **séquence d'opérations**: pour exécuter un comportement, un objet peut invoquer un ou plusieurs objets pour une séquence d'opérations. Parfois l'ordre de la séquence d'opérations demandées est très important, et toute permutation des opérations peut générer des comportements indésirables (le point a dans le modèle de fautes pour les logiciels).

5- **faute de sortie**: point a dans le modèle de fautes pour les machines à états finis.

6- **faute de transfert**: point b dans le modèles de fautes pour les machines à états finis.

7- **faute de transfert avec états supplémentaires**: point c dans le modèle de machines à états finis.

8- **faute de transitions supplémentaires ou manquantes**: point d dans le modèle de machines à états finis.

4. Sélection des tests à partir de spécifications orientées objets

Dans cette section, nous allons examiner les différents problèmes liés à la sélection des tests à partir des spécifications orientées objets. Nous identifierons les caractéristiques fondamentales de l'approche orientée objet, et les facteurs dont il faut tenir compte pour la phase de sélection des tests. Dans un premier temps, nous présenterons les différents aspects à tester, où chaque aspect sera traité de façon indépendante des autres. On examinera alors les difficultés et les restrictions à considérer pour chaque cas traité.

4.1. Sélection des tests pour le comportement d'un type d'objet

La spécification de chaque type d'objet définit les opérations qu'offre le type ainsi que les comportements associés aux invocations d'opérations. Nous faisons ici la restriction de ne considérer que les entités qui modélisent les objets et dont le comportement peut être modélisé par un automate d'états finis. Nous traiterons le cas des entités qui modélisent les relations à part. L'exemple de la figure 5, est un type d'objet qui offre deux opérations A et B. Son comportement, modélisé par un FSM, stipule que le type d'objet est initialement dans l'état e_1. A l'invocation de l'opération B, le type d'objet doit retourner le résultat o_2 et passer ensuite à l'état e_2. Une séquence d'invocations, formée de (B, A, B), entraînera le retour des résultats (o_2, o_3, $o_{2,}$) et le type d'objet passera ensuite à son état initial e_1. La transition qui relie les états e_5 et e_4 représente une invocation à un autre type d'objet pour l'opération C. Cette interaction peut ne pas être visible suivant le type de test adopté (par exemple, dans un test du type boite noire, cette transition ne peut pas être observée).

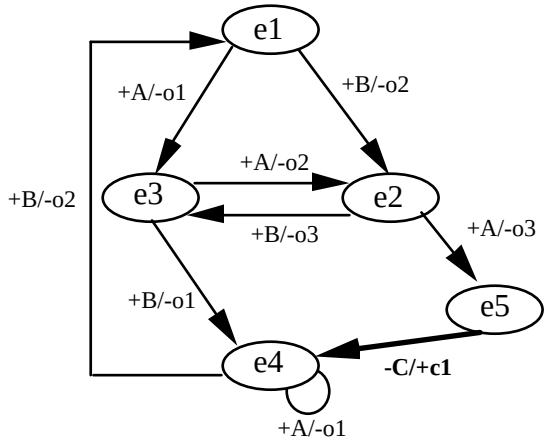


Figure 5: modélisation du comportement du type d'objet

La sélection des tests pour la couverture du comportement d'un type pourrait être faite suivant l'une des méthodes traditionnelles basées sur les FSMs (TT, DS, UIO, W). Le choix de la méthode de sélection, à ce niveau, dépend de la nature de l'automate (fortement connexe, complètement spécifié) et du niveau de détection que l'on veuille atteindre (par exemple, la méthode TT n'assure que la détection des erreurs de sortie). Nous verrons plus loin que d'autres facteurs peuvent influencer ce choix dépendamment du type de test adopté et de l'accessibilité au type d'objet sous test.

4.2. Test des paramètres

Le test des paramètres est un autre aspect à examiner. En effet, un type d'objet peut offrir des opérations avec des paramètres. Tester exhaustivement toutes les valeurs d'un paramètre peut prendre un temps très grand [Piat 80]. Devant l'impossibilité de faire un test exhaustif, il est préférable de déterminer des valeurs plus représentatives du paramètre à tester afin de pouvoir détecter le plus d'erreurs possibles.

Afin de déterminer des valeurs représentatives à tester, deux classes de valeurs sont à considérer: les valeurs valides et les valeurs invalides [Gama 91], [Vauc 91], [Vauc 91b]. Les valeurs valides représentent le cas où l'opération devrait être exécutée correctement, par contre les valeurs invalides correspondent aux cas contraires. Il est fort probable que l'implantation d'un type d'objet pourrait réagir face à des entrées invalides de façon inattendue d'où l'utilité du test du paramètre avec des valeurs invalides. La détermination des valeurs valides et invalides peut être faite de différentes manières [Gama 91] [Myer 79]. Une fois les valeurs représentatives pour chaque paramètre sont déterminées, la sélection des cas de test additionnels pour couvrir les paramètres d'une opération doit contenir le maximum possible de valeurs représentatives afin de minimiser le nombre total des cas de test. Ainsi des cas de test seront générés jusqu'à ce que toutes les valeurs représentatives valides soient couvertes. D'autres cas seront générés pour couvrir les valeurs représentatives invalides.

Les opérations peuvent avoir des paramètres dont les valeurs sont dépendantes entre elles. Dans ce cas là, la génération des cas de test présentée ci-dessus ne peut plus être appliquée. En effet, malgré que les cas de test sont générés sur la base des valeurs représentatives valides de chaque paramètre, ils peuvent ne tester que des comportements invalides, il en résulte une mauvaise couverture du comportement normal (la relation de dépendance entre les paramètres ne sera pas forcément couverte). Une méthode pour générer des cas de tests pour paramètres dépendants est proposée dans [Htite 93].

4.3. Test de conformité des instances d'un type d'objet.

Le type d'objet est le modèle conceptuel qui définit les attributs, les opérations et le comportement que toute instance de ce type doit avoir. Dans une implantation plusieurs objets peuvent avoir le même type. Ces objets doivent, à la fois, offrir les mêmes opérations et avoir les mêmes comportements. Le type de test qui nous intéresse ici a pour objectif de montrer que tous

les objets se comportent bien suivant leurs modèles conceptuels. Théoriquement cette vérification est simple; il suffit de générer une suite de test qui permettra de tester tout le comportement (et implicitement les opérations) du type d'objet (section 4.1). A cette suite on ajoute des cas de test pour couvrir la variation des paramètres en tenant compte des dépendances entre paramètres (section 4.2) , ensuite appliquer cette suite résultante sur toutes les instances du même type d'objet. Ces instances doivent se comporter toutes de manière similaire et en respectant les spécifications définies dans le type d'objet. Par similitude avec le domaine du "hardware", le test de conformité des instances dans une chaîne de production de cartes électroniques en série par exemple, consiste à faire subir les mêmes séquences de test à toute carte une fois ses composantes assemblées.

Le test de conformité de toutes les instances d'un type d'objet peut être long et coûteux, surtout si le nombre des instances est grand. Pour remédier à cet inconvénient, on adoptera la même solution que le test en "hardware", à savoir le test par échantillonnage. L'échantillonnage peut avoir deux formes:
1- **échantillonnage aléatoire**: où un certain nombre d'instances seront choisies de façon aléatoire pour subir le test de conformité des instances.
2- **échantillonnage par classe**: où une relation d'équivalence sera définie sur l'ensemble des instances d'un même type d'objet. Cette relation d'équivalence pourrait se baser sur plusieurs critères comme par exemple, la valeur d'un attribut. De chaque classe d'équivalence, ainsi définie, un candidat sera élu pour subir les tests. L'élection du candidat de la classe d'équivalence pourrait être aléatoire ou fixe (par exemple, la plus petite valeur d'un attribut dans la classe d'équivalence).

4.4. Test de l'héritage

L’héritage est une caractéristique fondamentale dans l’approche orientée objet. Un type d'objet peut hériter d'un ou de plusieurs autres types d’objet. Cet héritage se traduit, au niveau du type d'objet héritant, par la récupération de tous les attributs, interfaces et comportements des types d'objet hérités.

Sans perte de généralités, nous n'allons considérer que l'héritage du type enrichissement; cela consiste à ce que le type d'objet qui hérite offre, de façon intégrale, toutes les opérations et les comportements associés, définis dans les types d'objet desquels il hérite. Le type d'objet héritant peut aussi offrir, en plus, d'autres opérations et comportements qui lui sont propres (extension). Dans le cas où l'héritage est du type substitution (redéfinition des opérations et des

comportements hérités dans le type d'objet héritant), on pourrait ne pas considérer l'impact de l'héritage sur le test.

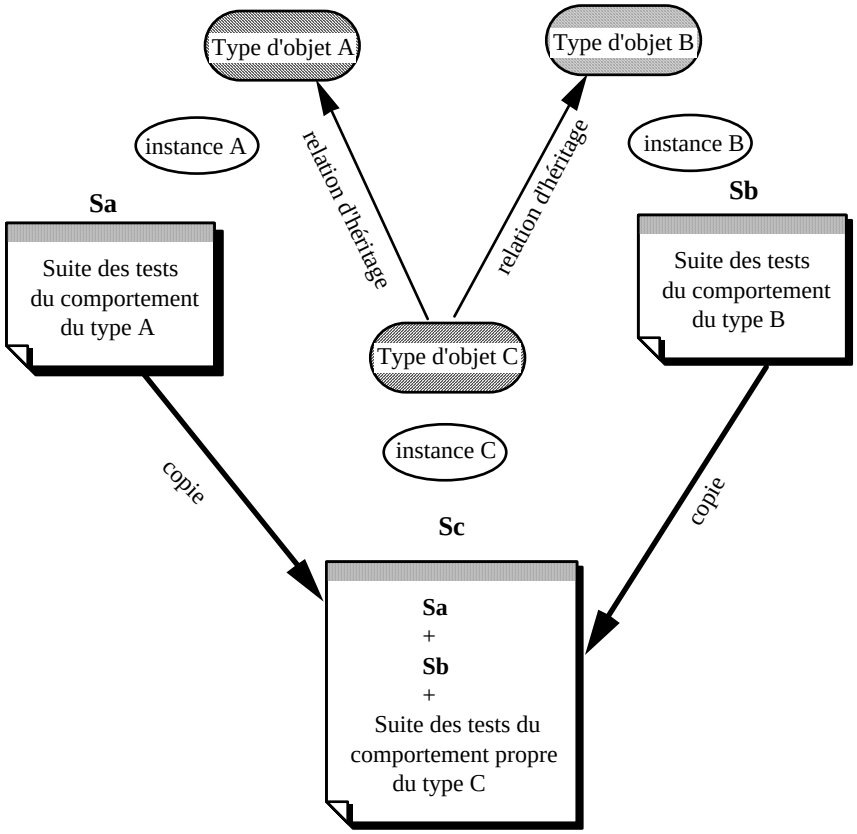


Figure 6: test de l'héritage

Dans ce contexte là, tester si l'héritage est bien implanté revient à s'assurer que le type d'objet héritant réagirait de la même façon que les types d'objet hérités. L'exemple de la figure 6, montre comment l'héritage pourrait être vérifié. Le type d'objet C hérite des deux types A et B. Le type C récupère ainsi toutes les opérations et les comportements des types A et B. Supposons que l'on dispose de deux suites de test (S_a et S_b) pour vérifier les comportements respectifs de deux instances de types A et B (section 4.1 et 4.2). Pour vérifier que l'héritage a été bien implanté, et en supposant qu'il n'y a pas de couplage entre les deux comportements, il suffirait d'exécuter les suites S_a et S_b sur une instance de type C; l'instance C devrait avoir la même réaction que les instances A (face à la suite S_a) et B (face à la suite S_b).

Il est évident de voir les facilités introduites par l'utilisation du mécanisme de l'héritage dans la spécification des protocoles de communication. Une spécification qui utilise l'héritage, permet de récupérer des comportements déjà définis, sans être obligée de les redéfinir de nouveau. L'utilisation de l'héritage à une autre répercussion avantageuse, mais cette fois-ci sur la sélection

des tests. Cet avantage réside dans la réutilisation des tests. Comme c'est illustré dans l'exemple de la figure 6, la suite S_C, pour tester le comportement du type d'objet C, est obtenue en réutilisant les deux suites S_a et S_b déjà générées; donc nul besoin de générer à nouveau les suites de tests pour les comportements hérités. La suite S_C contient aussi des séquences de test pour vérifier le comportement propre au type d'objet C.

4.5. Autres aspects à tester

Les attributs et les relations entre objets sont d'autres aspects à examiner. Nous les présenterons dans ce paragraphe afin d'identifier les difficultés liées à leurs tests.

4.5.1. Test des attributs

Un attribut d'un objet peut avoir plusieurs significations. Il pourrait spécifier une condition ou un état (par exemple, actif ou inactif), il peut être le moyen pour distinguer des objets du même type, comme il pourrait être la modélisation d'une relation entre objets. Nous ne considérons dans ce test que les attributs qui sont cachés et invisibles de l'extérieur de l'objet. Comme l'accès aux attributs qui nous intéressent n'est pas direct, on se penchera plutôt sur les effets de ces attributs afin de vérifier leurs implantations. Pour se faire, supposons que le comportement de l'objet dont on veut tester l'attribut est bien implanté. Plusieurs scénarios sont à envisager suivant ce que l'attribut peut représenter:

1) Attribut condition ou attribut état

Le test de ce type d'attribut pourrait être assuré par le comportement de l'objet. L'exécution de la partie du comportement (qui est supposé bien implanté) qui utilisera cet attribut condition ou état, et qui générera une sortie pourrait permettre de tester l'implantation de l'attribut. Cette vérification sera répétée plusieurs fois pour permettre de couvrir toutes les valeurs représentatives de l'attribut. Par exemple, si l'attribut représente les deux états inactif et actif, on exécutera le comportement pour vérifier que l'objet est bien dans l'état inactif, ensuite on invoquera l'objet pour l'opération ou les opérations qui vont le mener à l'état actif et enfin on exécutera de nouveau le comportement pour s'assurer que l'objet est bien, cette fois-ci, dans l'état actif. Il est à remarquer que le test de ces types d'attribut ne va pas engendrer des cas de test additionnels.

2) Attribut pour distinguer les instances

Soit l'exemple de la figure 7, qui est constitué de trois objets du même type (type A). Ces objets ont un attribut qui permet de les distinguer entre eux (A1, A2, A3). Soit aussi l'objet B qui pourrait invoquer un objet de type A. Au moment de l'invocation, on devrait préciser quelle instance serait invoquée (A1, A2 ou A3). Si l'objet B devrait faire ce choix de façon déterministe et si on suppose que les comportements des deux types d'objets A et B sont bien implantés, alors la couverture du comportement de l'objet B, qui concerne l'invocation de l'objet A, sera utilisée afin de tester l'attribut du type d'objet A.

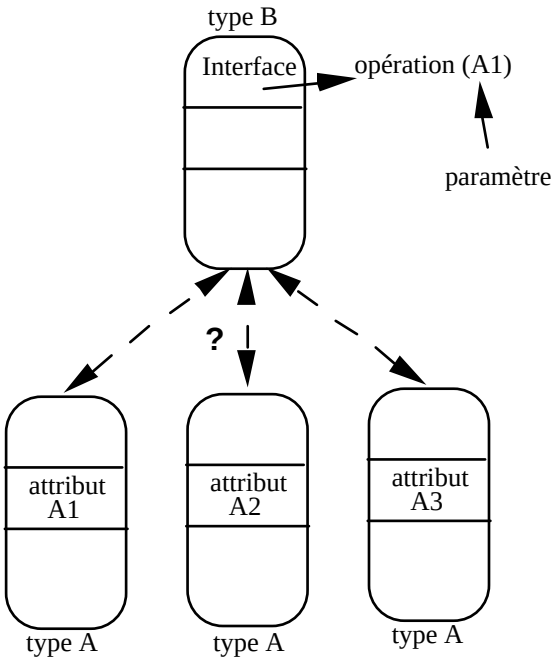


Figure 7: test des attributs

Parfois le choix de l'instance A à invoquer n'est qu'une conséquence d'une opération offerte par l'objet B. Cette opération lorsqu'elle est demandée, devrait contenir l'information sur l'instance du type A à invoquer. Cette information pourrait être passée comme paramètre de l'opération. Ce qui nous mène à conclure, sous l'hypothèse que les comportements des objets impliqués sont bien implantés, que la couverture des paramètres des opérations pourrait tester certains types d'attributs. Ce problème est identique au test des machines communicantes sous l'hypothèse d'observation partielle.

4.5.2. Test des relations

Le test des relations présente les mêmes difficultés que le test des attributs. La relation pourrait être modélisée par un attribut ou par un objet autonome. Cependant, le test de certaines relations qui ne représentent que la modélisation des dépendances des paramètres des opérations (figure 8), peut être relativement facile. Ainsi, les cas de tests additionnels pour vérifier si la dépendance des paramètres est bien respectée (section 4.2) revient en fait à s'assurer de l'existence de ces relations. Les autres cas de test avec des valeurs des paramètres qui ne respectent pas la dépendance des paramètres, tenteront de leurs coté de vérifier la inexistence de relations non spécifiées.

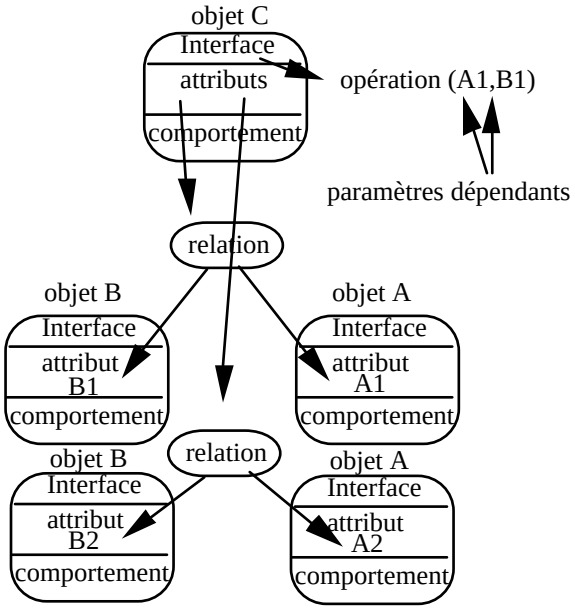


Figure 8: test des relations

En conclusion, et sous réserve que les comportements des objets impliqués sont bien implantés, le test des paramètres aussi bien que celui des dépendances entre les paramètres sont des instruments qui peuvent aider à tester certains types d'attributs et de relations.

5. Méthodologie de sélection des tests

Nous présenterons dans ce qui suit une méthodologie pour la sélection d'une suite de tests à partir d'une spécification orientée objet. Cette sélection suppose que l'on a juste une vue de service ce qui veut dire que l'exécution des tests ainsi que l'observation des réactions de l'implantation seront fait à partir des points d'accès au service; ce qui constitue donc un test du

type boîte noire("black-box"). Nous commencerons d'abord par définir notre modèle de test (figure 9) ainsi que les restrictions et les hypothèses à considérer.

Notre modèle orienté objet à tester est formé de plusieurs objets et relations où le comportement de chaque type d'objet est modélisé par une machine à états finis. L'exécution des tests à sélectionner vont se faire à partir des points de contrôle et d'observation (PCOs). En dehors de ces points, aucune observation n'est possible. Dans ce modèle, un PCO est un objet spécial de la spécification, cet objet joue le rôle d'interface entre l'environnement extérieur et les autres objets internes de la spécification.

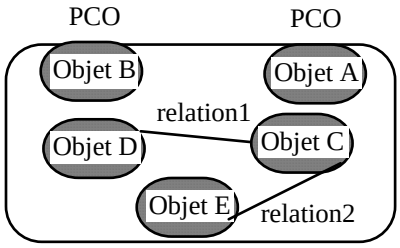


Figure 9: modèle d'implantation sous test (test "black-box")

Toutes les primitives de service (opérations) sont traitées par les objets PCOs, de même toute indication (sortie) destinée à l'environnement extérieur, se manifeste sur ces objets PCOs. La méthode de sélection que nous proposons a pour objectif de couvrir les classes de fautes que nous avons définies dans le modèle de fautes pour les spécifications orientées objets (section 3.3). Cela consiste à faire les sélections suivantes:
-cas de test pour vérifier les comportements des objets y compris les tests de conformité des instances, des attributs, des relations et de l'héritage si il y a lieu.
-cas de test pour vérifier la configuration .

5.1. Sélection des tests pour les comportements des objets

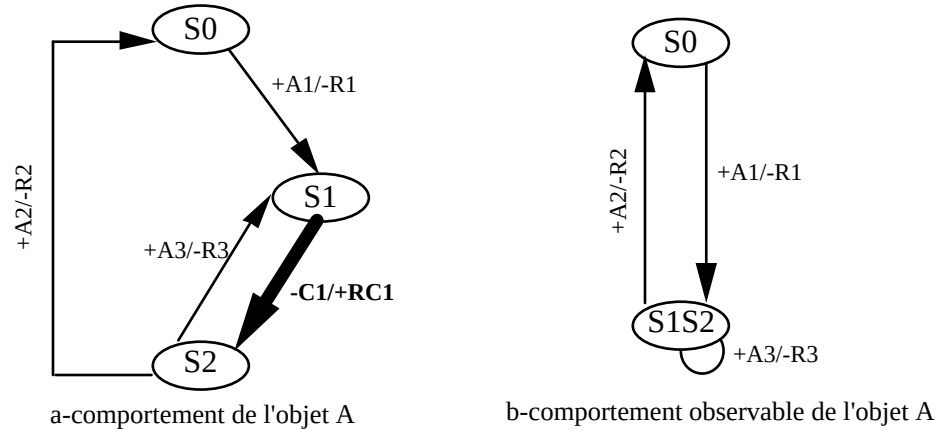
La sélection des tests pour les comportements des objets a pour but de vérifier la conformité des objets par rapport à leurs spécifications. Suivant la méthode de sélection de test adoptée, cela nécessite le parcours de toutes les transitions des machines des objets. La couverture du comportement n’est pas sans problème. En effet, seuls les comportements des objets directement accessibles (les objets PCOs) peuvent être couverts efficacement.

La figure 9 donne un exemple d’implantation. Elle se compose de cinq objets A, B, C, D et E. Seuls les objets A et B sont directement accessibles. La couverture du comportement de l’objet A ou de l'objet B peut se faire sans problème puisque toute entrée à l’implantation sera

immédiatement consommée par l'objet en question et toute sortie de l'objet destinée vers l'extérieur sera directement observable. Cependant, les transitions du comportement des objets A et B (les objets PCOs) qui représentent des invocations d'objets internes (les objets C, D et E) ne peuvent pas être directement observables. L'accès aux objets internes n'étant pas direct, leurs comportements ne peuvent pas être directement observables non plus.

a) Sélection des tests pour les objets directement accessibles

Soit la machine de la figure 10a qui modélise le comportement de l'objet A (l'objet A étant un objet PCO); supposons que cet objet offre les opérations A1, A2 et A3. Les sorties R1, R2 et R3 sont obtenues respectivement après l'invocation des opérations A1, A2 et A3. C1 représente une opération offerte par l'objet interne C; cette opération est invoquée par l'objet A sur l'objet C quand il est dans l'état S_1. Cette transition interne de l'objet A est invisible de l'extérieur. La machine de la figure 10b représente le comportement extérieur visible de l'objet A. C'est sur cette dernière machine que l'on pourrait faire la sélection des tests.



a-comportement de l'objet A

b-comportement observable de l'objet A

Figure 10: comportement complet et comportement observable de l'objet A

Les méthodes classiques de sélection des tests (TT, UIO, DS, W) peuvent s'appliquer ici sans problème car l'accès à l'objet sous test est direct. Cependant, la nature de la machine du comportement observable (complètement spécifié, fortement connexe) peut influencer le choix de la méthode de sélection. Une fois que le choix est fait sur la méthode de test, on sélectionne la suite de tests à laquelle on ajoutera des cas de test pour couvrir les paramètres (valeurs valides, valeurs invalides) en tenant compte des dépendances des paramètres. Dans le cas où un objet PCO est un héritier d'autres objets PCOs, on pourrait réutiliser les tests déjà générés pour ces derniers, et ainsi il ne restera plus qu'à sélectionner les cas de test pour le comportement propre et observable de l'objet sous test.

b) Sélection des tests pour les objets et les relations inaccessibles

Considérons l'objet C de notre modèle (figure 9). Cet objet C est amené à communiquer avec les autres objets du modèle (les objets A, B, D et E). La couverture des comportements des objets accessibles (les objets PCOs A et B) entraînera automatiquement la couverture d'une partie de comportement de l'objet C. Au cours de leurs exécutions, les objets A et B peuvent faire appel à l'objet C pour exécuter certains comportements. La figure 11 montre les différentes parties du comportement de l'objet C.

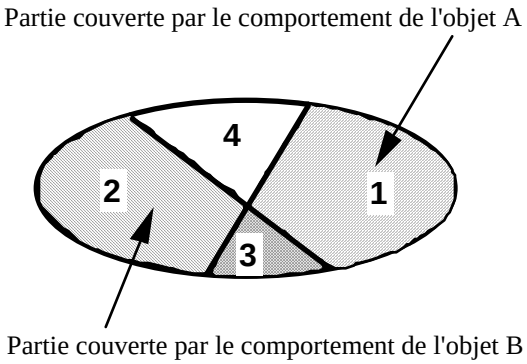


Figure 11: couverture du comportement de l'objet C inaccessible

Ainsi, la couverture des parties 1 et 2 est assurée respectivement par le comportement des objets A et B pris séparément. Quant à la couverture de la partie 3, qui représente l'intersection des parties 1 et 2, elle sera assurée à la fois par l'objet A et B. En fin, la couverture de la partie 4 peut être assurée par d'autres objets inaccessibles (par exemple, les objets D et E), ou en combinant les comportements des objets accessibles (comportement parallèle des deux objets A et B pour mener l'objet C à un état où il peut exécuter le comportement de la partie 4).

Deux cas sont cependant à considérer pour l'objet inaccessible sous test (figure 12):

1)-l'objet sous test peut être invoqué directement par un objet accessible (l'objet C, figure 12 a).
2)-l'objet est invoqué par l'objet accessible via un ou plusieurs objets inaccessibles (l'objet E, figure 12 b).

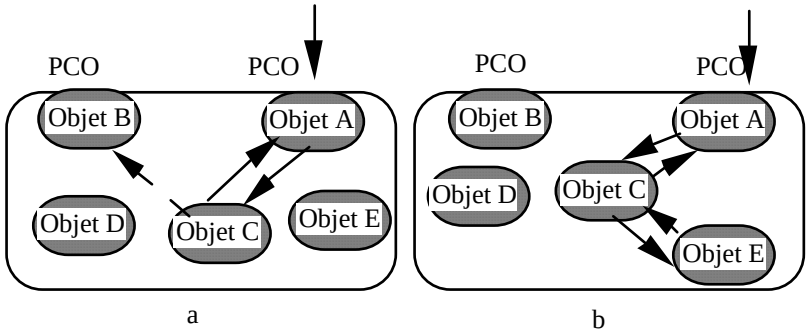


Figure 12: couverture des objets inaccessibles

Afin d'assurer la couverture du comportement de l'objet inaccessible, on est contraint, dans les deux cas, de poser l'hypothèse que les comportements des objets accessibles et des objets inaccessibles intermédiaires impliqués sont bien implantés, et sans cette hypothèse la couverture ne peut être assurée. L'idée consiste a voir, au niveau de chaque objet inaccessible, la conséquence des tests sélectionnés pour les objets PCOs et de compléter ces tests afin de couvrir la totalité des comportements des objets inaccessibles.

Comme cela a été mentionné à la section 4.5.2, certaines relations qui modélisent la dépendance des paramètres peuvent être implicitement testées en testant les paramètres. Pour le test des autres relations qui ne concernent pas la dépendance des paramètres, l'idée est d'activer à partir des objets PCOs le comportement qui ferait appel à ces relations et qui pourrait générer ensuite une sortie (sur un objet PCO) montrant ainsi l'existence et la bonne implantation de la relation. Évidemment, cette preuve suppose que les comportements des objets (accessibles et inaccessibles) impliqués sont bien implantés.

5.2. Sélection des tests pour la configuration

L’implantation sous test au cours de son traitement subit des changements dans sa configuration. Certains objets ou relations seront créés tandis que certains vont être détruits. Le test de la configuration a pour tâche de s’assurer que les créations et les destructions des objets ou des relations engendrées par le comportement spécifié d’un objet ont bien eu lieu et de façon correcte. La même hypothèse qui consiste à considérer que les comportements des objets sont bien implantés, reste applicable. L'idée pour assurer ce test est d'activer le comportement des objets PCOs qui vont faire appel aux objets ou relations qui sont détruits (pour vérifier qu'ils sont bien détruits) de même pour les objets et relations créés (pour s'assurer qu'ils sont bien créés).

6. Conclusion

Nous avons appliqué la méthologie proposée dans cet article sur l'exemple de la spécification de PCS "Personal Communication Service" [Desb 92]. Cette spécification est écrite dans le langage orienté objet MONDEL. PCS est un nouveau concept de communication basé sur les réseaux intelligents [Robr 91]. Il permet à ses usagers de gérer eux-mêmes la localisation et la configuration du service [Bala 90, Regn 90]. Cet exemple a permis de mettre en évidence le

problème de sélection de test à partir de spécification orientée objet en particulier le test des objets non accessibles, les relations et les attributs. En effet, pour pouvoir vérifier par exemple l'implantation des relations, l'idée a été de déclencher les comportements des objets accessibles qui réfèrent à ces relations et qui génèrent ensuite une sortie observable. Or, cela n'est pas toujours possible; on peut bien déclencher le comportement qui utilise la relation, mais rien ne garantit que l'implantation va générer une sortie observable. D'autre part nous avons posé l'hypothèse que le comportement des objets accessibles aussi bien que celui des objets intermédiaires étaient bien implantés. L'introduction de cette hypothèse pourrait être expliquée par l'observation partielle que nous avons dans notre système de test. En effet, comme nous avons seulement une vue de service, les interactions internes ne peuvent pas être directement observables. Cependant, si nous avons eu d'autres points d'observation sur les objets internes, on pourrait réduire l'impact des hypothèses posées.

Afin de pouvoir lever les restrictions posées, nous proposons dans cette conclusion quelques idées qui constituent une suite logique au travail que nous avons mené jusqu'à date et qui peuvent être des motivations pour des travaux futurs. Devant la complexité des aspects examinés et qui se résume à une observation partielle des comportements des objets internes, l'introduction d'instrumentations au niveau du type d'objet qui pourraient tester d'autres types d'objet de la spécification ne pourrait qu'augmenter l'observabilité du système sous test. Dans cette optique, on pourrait imaginer que chaque type d'objet serait formé de deux facettes. La première facette qui servirait à décrire le comportement de l'objet dans un environnement d'exécution normal; tandis que la seconde facette donnerait des moyens pour tester d'autres types d'objet de la spécification. Les objets ainsi conçus, peuvent se tester de façon mutuelle pour établir un verdict global de l'implantation. Dans le cas de présence d'anomalies, le verdict final sera propagé vers le système de test extérieur qui supervise l'implantation. L'avantage d'un tel système est de permettre à la fois de faire les tests avec plus d'observation et d'assurer le diagnostic pour la localisation des erreurs. Le parallélisme d'une part, l'instrumentation introduite d'autre part pourraient permettre d'effectuer les tests pendant le fonctionnement normal de l'implantation. Par exemple, on pourrait imaginer un réseau défectueux qui continue de fonctionner et dont les tests au niveau des objets tentent de localiser l'élément d'où provienne l'erreur pour l'éliminer.

Une autre difficulté qui n'a pas été rencontrée, mais qui peut bien se présenter, est le cas des comportement indéterministes. La sélection des tests à partir de machines à états finis indéterministe est un problème classique. G. Luo dans [Luo 92] a proposé une méthode pour permettre la sélection des tests à partir d'FSMs indéterministes. L'idée de cette méthode, est de faire subir des transformations à la machine indéterministe, sous certaines hypothèses, dans le

but de le rendre déterministe au sens d'entrées/sorties, mais ce but n'est pas toujours réalisable. Enfin, la sélection pourrait se faire à partir de la machine résultante des transformations.

Un autre point qui n'a pas été traité au cours de cet article est l'architecture du système de test. En effet, la coordination du déroulement des tests, à savoir la synchronisation des séquences de test, des différents testeurs, situés au niveau des objets PCOs, nécessiterait la mise en place d'un autre niveau de testeur, plus haut, qui permettrait de superviser et coordonner les testeurs du niveau plus bas. On trouvera dans [Meer 91] une présentation des problèmes liés à ce genre d'architecture de test distribué.

Réferences:

[Bala 90]:K. Balasubramanya and G. Rochlin, Universal Personal Telecommunications: Concepts and Requirements, IEEE, ICC'90, pp 0228-0232, 1990.

[Boch 90]: G. V. Bochmann, M. Barbeau, M. Erradi, L. Lecomte, P. Mondain-Monval and N. Williams: MONDEL, Object-Oriented databases Specification language, Technical Report from CRIM, Publication départementale #748, Département IRO, Université de Montréal, 1990.

[Boch 91]: G. V. Bochmann, M. Barbeau, A. Bean, M. Erradi, L. Lecomte and A. Liu: CRIM/BNR Projet : "Object-oriented databases" The Description of the Specification Language MONDEL V1, Centre de recherche informatique de Montréal , Janvier 1991.

[Boch 91a]:G.v. Bochmann, A. Das, R. Dssouli, M. Dubuc, A. Ghedamsi and G. Luo, "Fault models in testing (invited paper)", IFIP Transactions, Protocol Testing Systems IV (the Proceedings of IFIP TC6 Fourth International Workshop on Protocol Test Systems), Ed. by Jan Kroon, Rudolf J. Heijink and Ed Brinksma, 1992, North-Holland, pp.17--30.

[Boch 92]: G. V. Bochmann, S. Poirier and P. Mondain-Monval: Object-oriented design for distributed systems and OSI standards, Proc. of IFIP Int. Conf. on Upper Layer Protocols, Architectures and Applications, Vancouver, May 1992. A shorter version is also included in the proceedings of the Int. Workshop on ODP, Berlin, Oct. 1991.

[Chen 76]: P. P. Chen, The Entity-Relationship Model: Toward a Unified View of Data, ACM Trans. on Database Systems, Vol. 1, N 1, pp. 9-36, March 1976.

[Chow 78]: T. Chow:Testing Software Design Modeled by Finite-State Machines, IEEE Trans. Software Engineering, Vol.SE-4, pp.178-187, March 1978.

[Desb 92]: D. Desbiens: Modelization and Specification of the Personal Telecommunication Services, Thèse de Maîtrise, Département IRO, Université de Montréal, 1992.

[Gama 91]: P. Gamache: Générateur intelligent de tests adapté au domaine des protocoles de communication, Thèse de maîtrise, Département IRO, Université de Montréal, Mars 1991.

[Gone 70]: G.Gonenc: A Method for the Design of fault Detection Experiments, IEEE Transactions on Computer, Vol 19, N 6, pp. 551-558, 1970.

[Htite 93]: E. Houssain Htite: Géneration de tests pour le service de commuincation personnalisé, Thèse de Maîtrise, Département IRO, Université de Montréal, 1993.

[Koh 78]: Z.Kohavi: A switching and Finite Automata Theory, Mc Graw Hill, 1978.

[Luo 92]: G. Luo, G. V. Bochmann and A. Das: Test Generation for Concurrent Programs Modeled by Communicating Indeterministic Finite State Machines, Publication départementale #823, département IRO, Université de Montréal, Mai 1992.

[Mas 89]: G. Masini, A. Napoli, D. Colnet, D. Léonard et K. Tombre: Les langages à objets, interEditions, iia, Paris 1989.

[Meer 91]: J. de Meer, V. Heymer, J. Burmeister, R. Hirr and A. Rennoch: Distributed Testing, 4th International Workshop on Protocol Test Systems, Part IV, Oct 1991.

[Mond 90]: P. Mondain-Monval, G. V. Bochmann: An object-Oriented Software Design Methodology, Progress Report Document N 7 for CRIM/BNR Project, June 1990.

[Myer 79]: G. J. Meyers: The Art of Software Testing, Wiley-interscience publication, John Wiley and Sons, pp. 177, 1979.

[Nait 81]: S. Naito and M.Tsunoyama: Fault Detection for Sequential Machines by Transition Tours, Proceeding of the 11th IEEE fault Tolerant Computing Symposium, IEEE Computer Society Press, pp. 238-243, 1981.

[Piat 80]: L. M. Piatkowski: Remarks on the Feasibility of Validating and Testing ADCCP Implementations, Proc. of trends and applications symposium, (NBS), Gaithersburg, 1980.

[Regn 90]: J. Régnier and W. H. Cameron, Bell-Northern Research, Personal Communication Services: The New Pots, Globecom 90, San Diego, December 1990.

[Robr 91]: Richard B. Robrock: The intelligent Network-Changing the Face of Telecommunications, Proceedings of the IEEE, Vol 79 (1), pp. 7-20, January 1991.

[Sabn 88]: K. K. Sabnani and A. T. Dahbura: A protocol Test Generation Procedure, Computer Networks and ISDN Systems, Vol 17 (4), pp. 285-297. 1988.

[Sari 87]:B. Sarikaya, G.v. Bochmann and E. Cerny, "A Test Design Methodology for Protocol Testing", IEEE Trans. on SE, April 1987, pp.518-531.

[Ural 91]: H. Ural and Bo Yang, " A Test sequence selection method for protocol testing", IEEE Transaction on Communications, April 1991.

[Vauc 91]: J. Vaucher, G. v. Bochmann, B. Lefebvre, S. Desmarais and P. Gamache: Le projet MMS "L'informatique intelligente appliquée à l'implantation et au test de logiciels industriels", Intelligence Artificielle et Sciences Cognitives au Québec, Vol.3, N 3, pp.45-58, 1991.

[Vauc 91b]: J. Vaucher, G. v. Bochmann, B. Levebvre, S. Desmarais and P. Gamache: Le projet MMS "l'informatique artificielle appliquée à l'implantation et au test de logiciels industriels", présenté au Congrès de l'ACFAS, Sherbrooke, Québec, May 1991.