# Rapport Technique - Phase 2
# Étude et amélioration de la démarche de test liée à POK

Telecom ParisTech

22 octobre 2010

## Introduction

Ce rapport fait un court état des lieux et bilan sur la démarche de test et les premiers résultats obtenus en Phase 1 pour le support d'exécution POK. Sa fonction est de de faire le point sur les enjeux réels de l'utilisation de l'outil *xcov* pour évaluer la démarche de test d'un système comprenant une fraction importante de code non-fonctionnel (Nous utilison le terme non-fonctionnel pour désigner un code possédant des effets de bord dont l'étude est non trivialle et affecte le logiciel comme le matériel par exemple).

Le rapport est structuré de la manière suivante :

- Une présentation de l'usage de *xcov* dans une démarche assistée d'amélioration d'un ensemble de cas de test existants. Les tests ont pour cible l'intergiciel POK et les applications déployées par dessus ce dernier dans le cadre d'un logiciel suivant le standard architectural ARINC-653 du logiciel embarqué sur un calculateur avionique.
- L'inventaire des objectifs et des contraintes liées aux tests réalisés sur POK. L'identification des problèmes posés par la démarche de génération de code utilisée dans POK.
- Définition de classes des défauts de couverture détectés grâce à *xcov* dans les cas de test proposés en Phase 1.
- Une réflexion rapide sur une méthode de présentation de l'information de couverture tenant compte des spécificité du code testé dans POK : un code dont la fraction fonctionnelle ne représente pas nécessairement la majorité du système (i.e. il y a des branches qui ne sont activée qu'en présence d'événements inattendus ou très rares, tels que des fautes matérielles)

– La spécification d'un format associant l'information de couverture, sous forme d'intervalle de lignes et de taux, à des blocs de code (et non des lignes) avec une granularité allant de la fonction jusqu'aux structures de contrôles de type «if . . .then . . .else »...

# 1 Processus de test et d'amélioration de la couverture

L'évaluation de la couverture de code d'un système testé est requise du point de vue du processus DO-178C. Lors de la première phase du projet, nous avons établis un ensemble de cas de test servant à illustrer les fonctionnalités d'un support d'exécution partitionné, et mis à disposition en accès libre dans les «release» de POK.

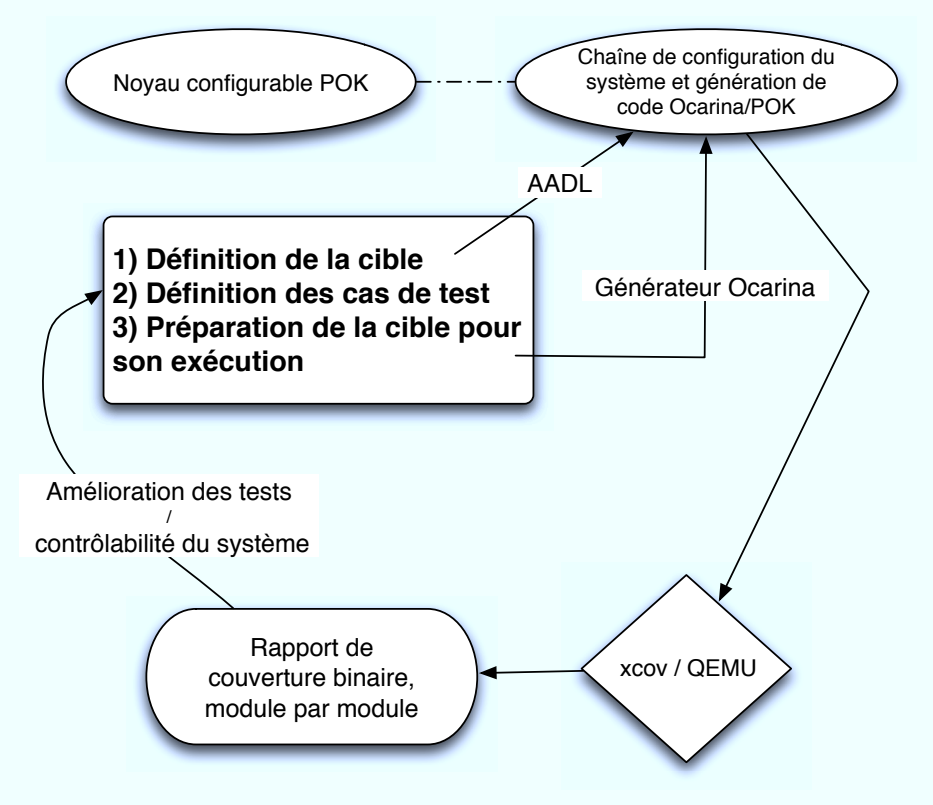


FIGURE 1 – Description de haut niveau du processus de test

Les caractéristiques de ce processus de test sont les suivantes :

– La préparation de la cible 1) repose sur des modèles AADL permettant de modéliser l'architecture de la cible sous formes de composants à l'interface explicitement définie
– L'assemblage de la cible en prévision de son exécution par *xcov* est assurée par un ensemble de scripts et la boîte à outils Ocarina. Ocarina regroupe des fonctions de vérification des modèles AADL, le processus de configuration de POK et de génération du code glue assurant l'interfaçage entre les différents composants métier de l'application et l'intergiciel. Le code source supporté est C et Ada pour la partie applicative et C pour l'intergiciel.
– L'exécution du test, la collecte de la trace d'exécution et la production du rapport de couverture sont assurées par l'outil xcov. L'enchaînement de ces étapes est défini dans un script inclus dans la chaîne de compilation de POK.
– L'interprétation du rapport consiste à déterminer les causes des défauts de couverture de code. Cette interprétation nécessite de connaitre la sémantique du code source cible. Cette interprétation doit permettre d'identifier les défauts de couverture faciles à éliminer, par exemple à travers l'altération des données d'entrée.
– Une fois que l'interprétation est faîte, les cas de test doivent pouvoir être aménagés en fonction de l'importance du défaut de couverture

Il est important d'augmenter au maximum la lisibilité des rapports à un haut niveau d'abstraction car le code exécuté est en partie un code généré dont la lecture est assez ardue.

# 2 Objectifs et contraintes de test : le cas du code généré à partir de modèles

Les objectifs du test sont de montrer que les chemins d'exécution des fonctionnalités de l'application ou des services de l'intergiciel n'engendrent pas de comportement aberrants.

La qualité des tests est déterminée par la qualité de l'oracle utilisé pour valider les exécutions de la cible, et la précision de l'outil utilisé pour déterminer le taux de couverture assuré par le test.

Lorsque le code source représente un code métier destiné à être intégré dans une architecture tel quel, l'évaluation de la couverture de code peut être faîte en isolation sous la forme d'un test unitaire.

En revanche, si le code est généré en fonction de modèles formels et modifie en profondeur une portion importante du système, alors il est nécessaire

d'adopter une vision assez proche du test d'intégration : construire la cible la plus complète possible et s'assurer que la cible tester sera effectivement déployée tel quelle lors du déploiement opérationnel.

Bien que le principe de la génération de code soit de construire une application dont le code est prouvé correcte par construction, il faut noter que le test reste un passage obligé. Il en découle que malgré le faible contrôle que l'on a sur les portions de code généré, il faut essayer de valider ces éléments du système dans une configuration similaire à celle qui sera déployée au final.

# 3 Sémantique pratique des défauts de couverture détectés sur POK

Nous avons pu identifier un certain nombre de cas distincts de défauts de couverture qu'il nous semble important de différencier.

**cas 0** La fonction ne fait pas partie du code effectivement appelé mais est inclues par le processus de génération de code.

**cas 1** Tout d'abord, il y a le cas classique d'un défaut de couverture du à une condition partielle couverte impliquant des données dépendant uniquement des paramètres d'entrée.

**intérêt** Du point de vue de la contrôlabilité, il suffit de contrôler les paramètres passés à la fonction pour améliorer le taux de couverture du test.

**cas 2** Le défaut de couverture dépend d'une condition liée à la valeur de retour d'une fonction implémentée par le support d'exécution ou l'application elle même. Dans ce cas, cette valeur peut être concrètement totalement incontrôlable.

**intérêt** En identifiant les structures de contrôles affectées par chacune de ces classes, il possible de trier les défaut de couverture en fonction du type de contrôle à réaliser sur le système à tester pour en améliorer le taux de couverture.

POK est un système partitionné qui possède des modes de fonctionnements déclenchés

# 4 Rapport de couverture hiérarchiques et Amélioration des tests

*xcov* possède différents formats de sortie pour la présentation des rapports de couverture. Nous avons tenté de définir les relations (en termes de schémas

de donnée) utilisées dans ces différentes sorties.

**asm :** une séquence liant ( un état de couverture, le code assembleur, un numéro de ligne ).

**xcov+ :** une séquence de structures de donnée liant l'état de couverture d'une ligne de code source à l'ensemble lignes de code assembleurs correspond à cette ligne de code source (ainsi que leur l'état de couverture).

**html+, xml :** même sémantique que les rapports xcov

Les rapports de *xcov*semblent être construits autour de la relation liant le code source au code objet ligne après ligne. Ce format possède l'avantage d'être le plus neutre possible. Il empêche de créer des dépendances majeures entre un langage spécifique de programmation et le contenu et la structure des rapports *xcov*.

Au contraire nous souhaitons profiter de la structure hiérarchique des grammaires (BNF par exemple d'un langage de programmation donné pour représenter les informations de couverture sous la forme d'intervalles et non de manière ponctuelle.

En particulier nous souhaitons pouvoir définir la profondeur d'une ligne de code non couverte dans l'équivalent d'un arbre syntaxique transcrivant de manière fidèle le fonctionnement des structure de contrôle (type if then else). Il est donc crucial que cette structure différencie les noeuds représentant un branchement dans l'exécution du code, des noeuds représentant une simple séquence d'actions (dépourvue de tout branchement. Puisqu'une structure de contrôle est défini sur de nombreuses lignes dont le code correspond à l'exécution des branches. Il est nécessaire de pouvoir représenter l'information de couverture comme l'association d'un taux de couverture et d'un intervalle de lignes de codes correspondant au noeud de la structure. L'intervalle permet de repérer dans le code d'origine la structure, le taux détermine la qualité de couverture de la structure complète.

La structure hiérarchique du schéma de donnée doit permettre d'associer au bloc des intervalles de lignes localisant le code « appartenant à la structure».

## Spécification

- Il existe un rapport par fichier de code source
- le rapport associé à un fichier est constitué d'une séquence de rapport élémentaires représentant l'information concernant une fonction.
- le rapport d'une fonction est constitué du résumé de couverture donnant la ligne de début et de fin de la fonction, puis le taux de ouverture

global. Le rapport contient aussi une séquence de blocs de trois types différents :
  - Structures de sélection (qui contient une condition, et des branches).
  - Blocs séquentiels purs.
  - Structure de boucle ou itération.
- un «noeud» de type branche peut contenir de manière récursive une séquence de ces blocs élémentaires
- Un rapport de fonction est une structure hiérarchique dont la profondeur est déterminer par le degré d'enfouissement des structures de sélection ou de boucle intégrées dans la fonction.

structures de contrôle imbriquées l programmation.