Génération de composants testables avec testeur distribué

Nicolas Belloir, Fabien Romeo

Université de Pau et des Pays de l'Adour
LIUPPA, B.P. 1155
64013 Pau CEDEX, France
{belloir, romeo }@univ-pau.fr

RÉSUMÉ. *La difficulté de valider l'intégration des composants dans une nouvelle architecture logicielle est une limitation importante pour le développement de l'ingénierie logicielle basée composant. Dans ce cadre, nous avons présenté dans le précédent Workshop OCM-SI la technologie BIT et la librairie BIT/J. Dans ce document nous présentons les travaux que nous avons mené pour étendre cette librairie afin, d'une part, qu'elle prenne en compte les composants sur étagère (COTS), et d'autre part de lui ajouter des capacités de management et de test à distance.*

ABSTRACT. *The difficulty to validate the component integration in a new architecture is a strong limitation to the development of the component-based software engineering. In this domain, during last OCM-SI workshop we presented the BIT technology and the BIT/J library. In this paper, we present our work to extend this library in two axis: firstly, to be able to run the BIT technology with COTS components and secondly to add some remote configuration and testing capabilities.*

MOTS-CLÉS : *Composant, Composabilité, Test Intégré, COTS.*

KEYWORDS: *Component, Componability, Built-in Test, COTS.*

1. La Technologie BIT et la librairie BIT/J

L'ingénierie logicielle basée composant (CBSE) [Szy99] apporte une réponse significative au développement de logiciels par réutilisation de code existant. Cette technologie récente présente encore cependant de nombreuses limitations à son plein essor [HCS01]. Notamment, il est difficile de valider l'intégration d'un composant dans une architecture logicielle. D'une part, la forme même des composants (par exemple l'aspect boîte noire ou l'accès uniquement possible via ses interfaces) rend leur test compliqué. D'autre part, il ne s'agit plus seulement de tester le composant lui-même (il est censé être validé par son fournisseur), mais aussi sa compatibilité avec l'architecture dans laquelle il doit s'intégrer. Il est donc fondamental d'accroître la *testabilité* des composants, élément essentiel pour leur qualité et leur fiabilité [VM95].

Parmi les travaux œuvrant dans ce sens, plusieurs approches ont proposé d'intégrer des tests ou des fonctionnalités de test au sein même des composants. Elles portent sur le test du composant lui-même (*self-test*) [Wan98], ou bien encore sur l'amélioration du processus de développement des jeux de test [JDT01]. [Gao02] propose la notion de *composant testable*, supportant le test à distance basé sur une interface de test générique et un modèle d'architecture spécifique. Notre approche est similaire à cette dernière, mais nous nous en démarquons en permettant la mise en œuvre de tests portant sur le comportement interne du composant (décrit sous forme de *Statecharts* [Har87]), basés sur les notions de *contrat* et de *qualité de service*.

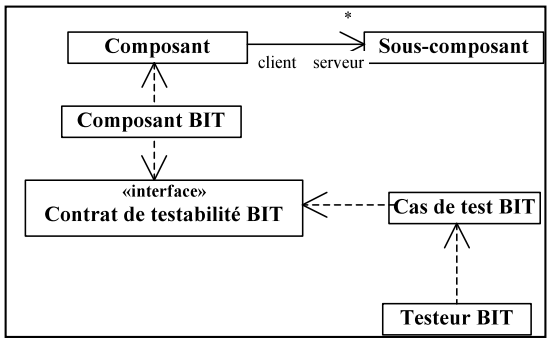

Figure 1 - Un exemple de mise en œuvre de la technologie BIT

Nous avons décrit dans [BBB03-1] la technologie BIT (*Built-In Test*) développée dans le cadre du projet européen *Component+* [1]. Parmi les résultats obtenus, nous avons présenté dans le précédent Workshop OCM-SI la librairie Java *BIT/J* mettant en œuvre cette technologie. La Figure 1 rappelle l'architecture que nous proposons. Un composant BIT est un composant logiciel qui, en plus de son interface initiale, propose une interface de test générique (contrat de testabilité BIT). Cette dernière

[1] Projet IST 1999-20162, pour plus d'informations, consulter http://www.component-plus.org

contient un ensemble de fonctionnalités de test permettant de manipuler le composant. Ces interfaces de test sont utilisées par des cas de test. Un testeur joue ces cas de test et affiche les résultats.

Nous avons depuis étendu les capacités de cette librairie suivant deux axes. Premièrement, nous avons amélioré les fonctionnalités de test afin de permettre leur utilisation avec des composants sur étagère (COTS), c'est à dire des composants non développés en interne et dont le code source n'est pas fourni. Ces particularités rendent encore plus complexe leur test car on ne peut accéder aux composants COTS qu'à travers leur interface. Notre librairie permet de spécifier le comportement qu'est censé avoir le composant COTS dans les interfaces de test, et de vérifier ensuite si ce comportement est bien implémenté. Ces travaux ont été présentés dans [BBB03-2]. Le second axe de recherche a porté sur la prise en compte de la manipulation des interfaces de configuration [Bos00] d'une part, et d'autre part sur la possibilité d'utiliser un testeur distribué pour tester le composant. En effet, l'ajout des opérations des interfaces de configuration dans le testeur permet de les manipuler dynamiquement et d'observer leur comportement. Enfin, nous avons doté la librairie d'un outil de génération automatique de code de test afin de la rendre plus efficace à utiliser, ce qui est important dans le cadre des développements rapides propres à la CBSE. Nous présentons dans la suite de ce document ces nouvelles fonctionnalités.

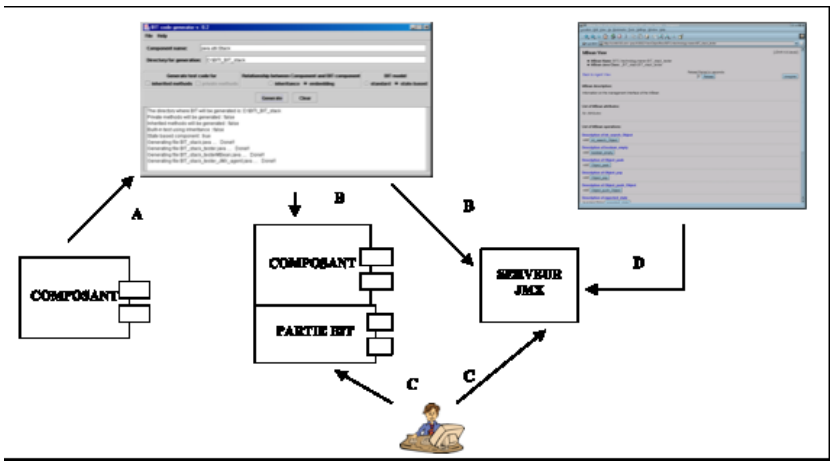

Figure 2 - création d'un composant BIT et de son testeur

2. L'outil de génération de BIT/J

La Figure 2 illustre le processus de génération du composant testable et des outils de management et de test distribué. Ces derniers utilisent la technologie JMX[2]

[2] Sun, *Java™ Management Extensions (JMX) 1.2.*, http://java.sun.com/products/JavaManagement/.

qui permet de gérer des composants à distance. Le processus de mise en œuvre de la technologie BIT se déroule globalement en quatre étapes.

La première étape consiste à lancer le générateur et à le configurer en lui spécifiant le composant à traiter et les options de génération (A sur la Figure 2). Trois options peuvent être sélectionnées :

- La première concerne le choix du modèle BIT (*standard* ou *basé état*). Le modèle BIT *standard* est l'approche la plus simple (interface de test minimale). Le modèle BIT *basé état* est le modèle implémentant l'interface permettant de décrire le comportement du composant à l'intérieur du composant BIT et de le tester.
- La seconde option concerne le type de relation entre le composant et sa version BIT. Deux types de relation sont possibles : le composant BIT peut hériter du composant ou l'encapsuler comme un attribut classique.
- La troisième option concerne la possibilité de prendre en compte ou non les opérations héritées du composant initial afin de les tester (cas où une opération fournie par l'interface est héritée).

La seconde étape (B sur la Figure 2) consiste à générer une partie du code du composant BIT, du testeur et des fichiers nécessaires à JMX.

Lors de la troisième étape, le développeur modifie et complète les fichiers générés (C sur la Figure 2). Le code généré pour le composant BIT contient un appel à toutes les opérations fournies par le composant initial. Le développeur peut compléter ce code en décrivant par exemple des éléments de test particulier. De plus, il peut ajouter des opérations de test spécifiques. Il décrit également lors de cette phase quelles opérations seront manipulables via l'interface JMX.

La quatrième étape (D sur la Figure 2) est l'étape d'exécution des tests. Pour ce faire, le développeur lance le serveur JMX et se connecte au composant BIT via un simple navigateur Web. Il peut le faire localement ou depuis une machine distante (applications embarquées ou distribuées).

3. Le testeur distribué basé sur JMX

Les évolutions du testeurs ont été dictées par plusieurs raisons. D'une part, dans la première version de la librairie, le testeur devait être codé manuellement et il ne présentait pas d'interface conviviale. D'autre part, les composants sont parfois déployés sur des systèmes distants (système cible différent du système d'intégration par exemple). Dans ce cas, la consultation des résultats des tests peut se révéler difficile voir impossible. Nous avons traité ces deux problèmes grâce à la technologie JMX. Elle permet la manipulation de composant distants par un navigateur Web.

Le testeur distribué basé sur JMX se présente donc comme une page Web. Chaque opération de test du composant BIT y est accessible et permet de lancer son exécution. Cela provoque l'affichage d'une nouvelle page Web indiquant le résultat de l'opération lancée.

Nous avons présenté dans ce document les nouvelles fonctionnalités de la librairie BIT/J. Pour plus de renseignements, vous pouvez consulter l'URL suivante à partir de laquelle il est possible de télécharger la librairie, son manuel utilisateur ainsi que l'exemple complet présenté ici :

http://liuppa.univ-pau.fr/themes/aoc/aoc/bitj.php

4. Bibliographie

[BBB03-1] Belloir N., Bruel J.M. et Barbier F., *Test intégré dans les composants logiciels*, numéro spécial de la revue L'Objet, à paraître.

[BBB03-2] Barbier F., Belloir N. et Bruel J.M., *Incorporation of Tests into Software Components*, proceedings of The 2nd International Conference on COTS-Based Software Systems, Ottawa, Canada, LNCS 2580, Springer, pp. 25-35, February 10-12, 2003

[Bos00] Bosch, J., *Design and Use of Software Architectures* – Adopting and Evolving a Product-Line Approach, Addison-Wesley, May 2000.

[Gao02] Gao, J., Gupta K., Gupta S.,Shim, S.: On Building Testable Software Components. proceedings of The 1st International Conference on COTS-Based Software Systems, Orlando, USA, LNCS 2255, Springer, pp 108-121, February 4-6, 2002.

[Har87] Harel D., *Statecharts : a visual formalism for complex systems* , Science of Computer Programming, Vol 8, pp 231-274, 1987.

[HCS01] High Confidence Software and Systems Coordinating Group, *High Confidence Software and Systems Research Needs*, Interagency Working Group on Information Technology Research and Development, January 2001.

[JDT01] Jézéquel J.-M., Deveaux D., Le Traon Y., *Reliable Objects : Lightweight Testing for OO Languages* , IEEE Software, Juillet / Août 2001.

[VM95] Voas, J. M. and Miller, K. W.: *Software Testability: The New Verification*. IEEE Software, Vol. 12 (3). IEEE Computer Society Press (1995) 17-28.

[Szy99] Szyperski C., *Component Software: Beyond Object-Orientded Programming*, Addison-Wesley, 1999.

[Wan98] Wang Y., King G., Patel D., Court I., Staples G., Ross M. and Patel S., *On Built-in Test and Reuse in Object-Oriented Programming* , ACM Software Engineering Notes, Vol. 23, No.4, pp.60-64. 1998.