Testing para Desarrollos Basados en Componentes

Elsa Estévez Marisa Sánchez

Departamento de Ciencias de Computación
Universidad Nacional del Sur
Av. Alem 1253 – (8000) Bahía Blanca
Argentina

e-mail: {ece,mas}@cs.uns.edu.ar

Palabras claves: Testing de integración, desarrollo basado en componentes, especificaciones formales

1 Introducción

La ingeniería de software basada en componentes (component-based software engineering) cumple un rol muy importante en el desarrollo de software, por ejemplo, considerar las infraestructuras de componentes que se ofrecen en el mercado como Enterprise JavaBeans [MH99], COM+ [Rog97], CORBA [Sei99].

Un sistema basado en componentes se construye integrando módulos que encapsulan datos y funcionalidad configurables mediante parámetros en tiempo de ejecución. La European Conference of Object-Oriented Programming de 1996 (ECOOP96) ha definido componente como *una unidad de composición que posee una interface especificada mediante contratos y dependencias de contexto explícitas. Una componente de software es entregada en forma independiente y sujeta a integración por terceras partes.*

La composición es la técnica clave a partir de la cual se construye el software e introduce nuevos aspectos a considerar. El usuario está interesado en ensamblar componentes, y por lo tanto, necesita analizar las consecuencias de la integración. Las componentes que funcionan correctamente en forma individual pueden dejar de hacerlo una vez que son integradas. Es por esto que para el desarrollo de software basado en componentes el testing de integración cumple un rol destacado. Debido a que en la mayoría de los casos el usuario no tiene acceso al código, no se pueden aplicar muchas de las técnicas de testing basadas en el análisis del mismo. Sólo

podemos analizar las propiedades de lo ensamblado a partir de la representación de "caja negra" de las componentes y de la información ofrecida por el proveedor. Generalmente, el proveedor describe el comportamiento de una componente especificando una lista de nombre de operaciones y sus correspondientes parámetros de entrada y salida, una descripción textual informal sobre la funcionalidad de la misma, una descripción informal sobre el contexto en el cual opera la componente (plataforma de hardware y sistema operativo), datos de performance y disponibilidad para una ejecución típica. Sin embargo, para facilitar el testing de una solución completa se necesitan descripciones más detalladas y rigurosas.

2 Línea de investigación

En un desarrollo de software basado en componentes es de vital importancia la identificación de dependencias entre las distintas piezas del sistema. A modo de ejemplo, existen dependencias de flujo entre productores y consumidores de recursos, dependencias en los protocolos de coordinación para recursos compartidos y dependencias temporales. Las técnicas de especificación actuales no proveen una forma adecuada para la representación de las mismas.

Debido a que las componentes se desarrollan en forma independiente resulta necesario realizar hipótesis sobre el ambiente en el cual se integrarán. El problema es que estas hipótesis no son, en general, especificadas adecuadamente y las especificaciones de componentes se restringen a definir una interface. En este sentido, la interface constituye el contrato que establece qué es lo que el cliente necesita hacer para utilizarla, y qué debe implementar el proveedor para satisfacer los requerimientos prometidos por la interface.

A nivel de operaciones, el método de especificación de contratos más conocido se basa en describir pre y post-condiciones para la operación. El cliente debe satisfacer la precondición antes de invocar una operación, y el proveedor asume que la pre-condición siempre se satisface cada vez que se invoca a la operación. La idea es simple y práctica pero existen muchos aspectos que no son cubiertos [Szy98] y dificultan la identificación de dependencias. Por ejemplo, el uso del mecanismo de *callback* (un procedimiento que es pasado a una librería) introduce dependencias difíciles de detectar. La librería invoca al callback y los estados intermedios de la librería se hacen accesibles a los clientes lo cual puede conducir a situaciones indeseables [Szy98]. En general, el método de especificar pre y post condiciones resulta insuficiente para describir las dependencias, y durante el desarrollo se pueden introducir errores. Debido a que las dependencias no fueron especificadas adecuadamente, estos errores resultan difíciles de detectar durante el testing de integración.

Una especificación constituye una fuente de información a partir de la cual se pueden generar casos de test. Varios autores proponen técnicas que integran especificaciones formales y testing, de forma que el testing puede ubicarse dentro de una base teórica, y así proveer una respuesta más convincente al problema de detectar errores (por ejemplo [BGM91], [CR96], [SC96]). Además, el testing basado en especificaciones formales tiene el potencial de ser automatizado. Por estas razones se está estudiando cómo determinar qué información es relevante para definir tests de integración que consideren las dependencias mencionadas previamente; describir cómo esta información puede especificarse usando una notación formal; y finalmente, indicar cómo derivar casos de test a partir de dichas especificaciones.

Referencias

[BGM91] G. Bernot, M.-C. Gaudel, and B. Marre. Software testing based on formal specifications: a theory and a tool. *Software Engineering Journal*, 6(6):387–405, November 1991.

[CR96] Juei Chang and Debra J. Richardson. Structural Specification-based Testing with ADL. In *Proceeding of the 1996 Internationl Symposium on Software Testing and Analysis*, pages 62–70. ACM, 1996.

[MH99] R. Monson-Haefel. *Enterprise JavaBeans*. O'Reilly Press, 1999.

[Rog97] D. Rogerson. *Inside COM: Microsoft's Component Object Model*. Microsoft Press, 1997.

[SC96] Phil Stocks and David Carrington. A Framework for Specification-Based Testing. *IEEE Transactions on Software Engineering*, 22(11):777–793, November 1996.

[Sei99] J. Seigel. What's comming in corba 3? *Java Developers Journal*, October 1999.

[Szy98] Clemens A. Szyperski. *Component Software*. Addison-Wesley, 1998.