Aplicación del modelo de verificación de componentes Itacio a una teoría sobre la fiabilidad de los componentes software

Agustín Cernuda del Río **Jose Emilio Labra Gayo** **Juan Manuel Cueva Lovelle**
Universidad de Oviedo - Departamento de Informática
Facultad de Ciencias - C/ Calvo Sotelo, S/N - Oviedo, España, 33007
{guti, labra, cueva}@lsi.uniovi.es

RESUMEN

Existen diversas técnicas de verificación y validación que persiguen comprobar la corrección de un sistema software a priori (sin ejecutarlo). Como contrapartida, la aplicación de estas técnicas puede requerir una importante base matemática, y además es frecuente que su aplicabilidad esté limitada a ciertos niveles de abstracción. Aun cuando en ciertas técnicas formales esto pueda no ser así, la industria del software suele tener esta percepción, errónea o no, en términos generales.

La noción (abierta) de "componente software" puede jugar un papel importante en la verificación de la calidad; un modelo de componentes simple y versátil permite incorporar conocimiento al proceso de desarrollo, y la adecuación de los componentes entre sí puede verificarse incluso sin necesidad de construir el programa. Además, puede aplicarse en muy diversos niveles de abstracción de la producción de software, lo que presenta ventajas frente a las técnicas mencionadas.

El modelo de componentes Itacio se ha desarrollado teniendo muy presentes estas restricciones y objetivos. Su versatilidad queda probada por haberse aplicado a la verificación de muy diversos tipos de propiedades de los componentes; entre otros usos, como caso particular es posible aplicar Itacio a un modelo de fiabilidad de los componentes, propuesto por Hamlet, Woit y Mason.

Palabras clave: Componente software, verificación, validación, modelo de componentes, fiabilidad, ingeniería del software, proceso de desarrollo del software.

Introducción

En los últimos años ha tenido un cierto impacto la noción de "componente software", que habitualmente [6] se asocia con la idea de un elemento ya compilado (librería, módulo o similar), generalmente desarrollado por terceras partes, y que se adquiere con el fin principal de ahorrar tiempo de desarrollo y mejorar la calidad. No obstante, esto no siempre es así. Independientemente de la calidad de un componente individual, se ha demostrado que la combinación de varios componentes puede generar problemas nuevos de tipo *emergente* que no resultan de la mera adición de errores [3].

Además de esto, no suele hacerse hincapié en la necesidad de métodos que permitan verificar que realmente al utilizar un componente su entorno cumple todas las condiciones que cada fabricante ha asumido como nominales. Estos requisitos suelen residir en documentación destinada a lectores humanos. En general, los denominados "modelos de componentes" comerciales, tales como COM, CORBA o JavaBeans, suelen servir como simple apoyo para resolver problemas de interoperabilidad entre plataformas o similares. Las precondiciones y postcondiciones, (si es que se utilizan) son, en general, instrucciones más ejecutables que declarativas; en la práctica, rara vez se piensa en analizarlas estáticamente.

El modelo de componentes Itacio

En esta línea, se ha desarrollado un modelo de componentes [1, 2, 5], denominado **Itacio**, que pretende incorporar las ideas esbozadas en el párrafo anterior. El término "componente" adquiere aquí una interpretación muy abierta. Itacio se ciñe a unas restricciones básicas:

- **Simplicidad**, gracias a la cual se pretende conseguir objetivos como los siguientes:
  - **Fácil comprensión del modelo**. Se pretende que su aplicación práctica resulte atractiva, disminuyendo los costes de aprendizaje y posibilitando de hecho que se implante en el proceso de desarrollo. Esto no ha ocurrido, en general, con otros métodos formales.

    - **Flexibilidad**. Un modelo simple puede aplicarse a diversos propósitos y niveles de abstracción. Esto mejora notablemente la rentabilidad de la adopción de la técnica.
- **Viabilidad técnica**. Se pretende que el modelo se pueda implementar con tecnologías existentes y a un coste razonable, sorteando, por ejemplo, problemas de computabilidad o decidibilidad. En nuestro caso, con recursos mínimos se han podido implementar prototipos funcionales, lo que mueve al optimismo respecto a su viabilidad.

La noción central del modelo es la de *componente* (término utilizado de manera heterodoxa). Ya se ha adelantado que en Itacio no es necesariamente un módulo ejecutable; es simplemente un artefacto software con fronteras definidas. Por tanto, pueden ser componentes las instrucciones individuales, los contratos de reutilización, las clases/objetos, o muchos otros elementos software a diferentes niveles de abstracción.

La frontera de un componente es "permeable"; presenta puntos de conexión que pueden ser *fuentes* ("salidas" del componente) o *sumideros* ("entradas" en el componente). El componente contiene, además, *expresiones restrictivas* que hacen referencia a las fuentes y sumideros. Estas expresiones pueden ser *requisitos* (plantean exigencias sobre los sumideros) o *garantías* (afirmaciones respecto a las fuentes). La creación de un *sistema* exige la composición de varios de estos componentes de modo que una fuente de un componente se conecta con un sumidero de otro componente.

Dado uno de estos sistemas, el proceso de verificación comienza con la generación de una *base de conocimientos* que lo describa, combinando (como se detalla en [1]) las expresiones restrictivas de manera que la topología del sistema esté implícita en esa base de conocimientos. En Itacio, las expresiones restrictivas se denotan utilizando lógica de primer orden en forma de cláusulas Horn, y se implementan sobre un lenguaje de Programación Lógica con Restricciones (CLP, de *Constraint Logic Programming*). Creada la base de conocimientos, sólo resta verificar que todas las exigencias de todos los componentes se cumplen; un sistema de inferencia convencional puede realizar deducciones sobre esta base de conocimientos con garantías de compleción. Si una restricción no se satisface, el sistema de inferencias puede indicar exactamente qué es lo que falla, dónde, y dar explicaciones acerca del problema. Con estas expresiones resulta bastante sencillo, por ejemplo, modelar el sistema de verificación de signaturas tradicional (métodos, parámetros, tipos).

Para la aplicación práctica del modelo en diversos ámbitos, se realiza un proceso de *instanciación* del mismo, que consiste en identificar los conceptos del dominio del problema con los elementos básicos de Itacio (componentes, fuentes, sumideros). Pueden verse diferentes ejemplos concretos de aplicación en [2].

Modelo de fiabilidad de Hamlet et al

Además de la habitual verificación de signaturas, es frecuente que se plantee la necesidad de pensar en otras características de los componentes, como pueden ser restricciones de eficiencia o de fiabilidad (entendiendo la fiabilidad como la tasa de fallos). En ciertas aplicaciones se plantea la necesidad de que la composición de componentes software pueda realizarse bajo condiciones de fiabilidad conocidas y predecibles. En este marco se puede encuadrar el trabajo de Hamlet et al [4]. Estos autores presentan una teoría de la fiabilidad basada en componentes; aquí se propone una representación de dicha teoría en el modelo Itacio, lo que constituye un caso más de aplicación de Itacio en un campo diferente (el modelado de la fiabilidad de un sistema).

En este caso, la aplicación de Itacio debe entenderse como un ejercicio demostrativo y no como un caso real de uso, puesto que el propio trabajo de investigación de Hamlet et al aún no ha finalizado. Se pretende sólo comprobar la versatilidad de Itacio, al ser posible aplicarlo a este problema sin grandes dificultades.

Una teoría de la fiabilidad de un sistema basada en componentes

Lo que pretende el modelo de Hamlet et al es que los desarrolladores de los componentes puedan realizar mediciones, que son utilizadas después por quienes diseñan sistemas basados en ellos para calcular (antes de la implementación o las pruebas) la fiabilidad del sistema resultante. Hamlet et al proponen el uso de diversas hojas de datos que describen un componente. El *perfil operacional* caracteriza (en términos de probabilidad de valores) el dominio en el que se llevaron a cabo las pruebas del componente, o bien el dominio en el que se utilizará. Los datos de fiabilidad utilizan como parámetro el perfil operacional concreto. Por último, existe también una correspondencia entre el perfil de entrada y el de salida. Por razones de espacio, resulta imposible repetir aquí la teoría de Hamlet et al, pero puede consultarse en [4].

Aplicación de Itacio al modelo de fiabilidad

Supóngase que se quiere aplicar el modelo Itacio para describir y verificar restricciones de fiabilidad de los componentes. En este apartado se aplicará un prototipo de Itacio (denominado Itacio-XDB [5]) al mismo ejemplo presentado en [4].

El primer paso es la instanciación del modelo en el dominio del problema. La asimilación del concepto de componente de Itacio parece clara, ya que en el modelo de Hamlet también se hace referencia directa a componentes software; en este caso, la correspondencia es directa. Respecto a la frontera de los componentes (sus fuentes y sumideros) hay varias opciones, pero aquí se ha optado por que los sumideros y fuentes representen a los subdominios de entrada y salida, es decir, cada sumidero representa un subdominio de entrada y cada fuente un subdominio de salida. Además de los dominios de entrada y de salida de cada componente, también se incorpora una salida de cada uno de ellos, que indica su tasa de fallos dado el perfil de funcionamiento en el que se hallan inscritos. Por último, es necesario un componente que suministre los dominios de entrada al componente A, y también habrá sendos componentes de verificación cuya única misión es recibir el valor de la tasa de fallos y establecer los requisitos deseados.

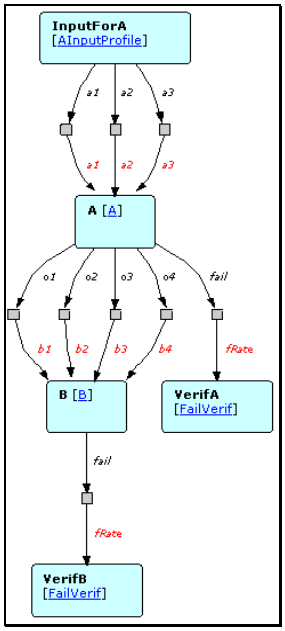

Fig. 1. Modelo de fiabilidad del ejemplo.

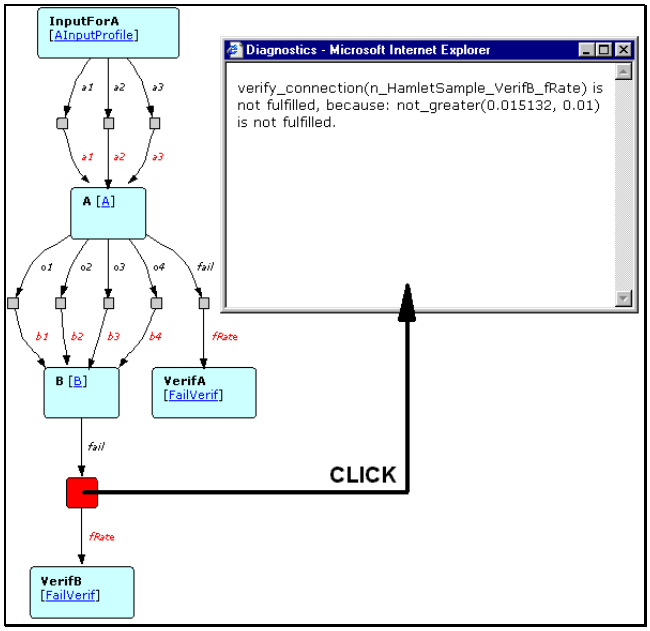

Fig. 2. Modelo de fiabilidad del ejemplo, con restricciones de fiabilidad más severas (capturas de pantalla del prototipo Itacio-XDB).

De este modo, se obtiene la representación gráfica de la Fig. 1. Se aprecian los componentes A y B, el componente que representa el perfil operacional en el que se va a utilizar A, los componentes de verificación (VerifA y VerifB) que comprueban que la tasa de fallos de cada componente no supere la deseada, y los diferentes sumideros y fuentes (excepto las de tasa de fallos) representan la información sobre los subdominios de entrada y salida (no las entradas y salidas individuales).

Las garantías de InputForA establecen la información sobre el perfil operacional de A. Las garantías de A, a su vez, realizan los cálculos de transformación de perfiles, así como el cálculo de la tasa de fallos del componente a partir del perfil operacional de entrada. Las expresiones restrictivas que describen este componente A no son más que cláusulas Horn que reflejan esta información. Por ejemplo, una de las garantías de A es[1]:

```
profile_map($o2$, X) :- profile_map($a1$, A1), profile_map($a2$, A2), profile_map($a3$, A3),
                        X is A1 * 0.003 + A2 * 1.0 + A3 * 0.002
```

En este caso, el sistema es correcto. La tasa máxima de fallos admisible se representa como un parámetro de entrada (sumidero) para los componentes de tipo FailVerif; en este caso, tanto VerifA como VerifB tienen la

[1] En el prototipo Itacio-XDB las fuentes y sumideros se representan en las cláusulas Horn encerradas entre símbolos de dólar. De este modo, un simple proceso de sustitución de cadenas puede generar la base de conocimientos.

tasa máxima de fallos fijada en 0.02. Pero si se modifica el parámetro `maxFailRate` de VerifB para plantear mayores exigencias, disminuyendo dicha tasa máxima admisible hasta 0.01, el sistema detectará que, bajo el perfil operacional de A y dada la transformación de perfil que A realiza, el perfil operacional en el que se encuentra B no permite garantizar una tasa de fallos por debajo de 0.01. En el prototipo Itacio-XDB, el error se señala mediante un cuadrado rojo de mayor tamaño (Fig. 2), y pulsando sobre él el usuario puede obtener una explicación sobre el problema, que consiste en que la tasa de error de B es en realidad 0.015132.

Más allá de la composición básica

Las operaciones descritas hasta ahora se corresponden con la llamada **composición básica** del modelo de Hamlet. Hay problemas adicionales a tener en cuenta, sobre todo cuando entra en juego la "lógica de interconexión" (*glue logic*). Construcciones del tipo **if** y **while** producen sucesivas particiones de los dominios involucrados; los autores del modelo proponen utilizar las probabilidades de las diferentes expresiones condicionales, que jugarían un papel parecido al de los perfiles de transformación. En términos generales, este tipo de análisis no es trivial, aunque el tratamiento de cláusulas **if** ya se contempla en el modelo. Los bucles son construcciones problemáticas en el análisis de programas, y en el modelo de Hamlet plantean especiales dificultades; Hamlet et al son optimistas en el caso de su modelo, pero no ofrecen una solución definitiva.

Conclusiones y trabajo futuro

El modelo Itacio se suponía lo bastante flexible para encajar con diferentes nociones de componente y expresar muy diversos tipos de restricciones, aportando una forma viable (en términos de implementación y uso) para verificar combinaciones de componentes software. Esto se ha ido comprobando al aplicar este modelo a diversos ámbitos, dado que Itacio no se había diseñado para ser aplicado en estos campos concretos. En este caso (aun cuando el modelo de Hamlet no está completo) se aprecia nuevamente que en principio Itacio es aplicable a la verificación de la fiabilidad.

Algunas vías futuras de actuación del proyecto Itacio podrían plantear la incorporación de técnicas de Ingeniería del Conocimiento para guiar la redacción de expresiones restrictivas, de modo que den lugar a bases de conocimiento más homogéneas, correctas y eficientes. También podría ser de interés el estudio de técnicas que permitan relacionar las expresiones restrictivas con la implementación real de los componentes y verificar que esta implementación se ajusta a dichas expresiones restrictivas. Resultaría especialmente atractivo avanzar en la posibilidad de realizar inferencias sobre sistemas inacabados; esto sugiere el estudio de técnicas de inferencia que deduzcan qué restricciones deberían cumplir los componentes que faltan, lo que podría dar lugar incluso a un **sistema de ayuda al diseño**, que sugeriría componentes apropiados para completar una configuración que se haya determinado sólo parcialmente.

Bibliografía

1. Cernuda, A., Labra, J. E., Cueva, J. M. *A Model for Integrating Knowledge into Component-Based Software Development*. Actas KM - SOCO 2001, 26-29 de Junio de 2001, Paisley (Escocia). ICSC Academic Press, ISBN 3-906454-27-4.
2. Cernuda, A., Labra, J. E., Cueva, J. M. Diversas publicaciones sobre itacio: disponibles en `http://www.agustincernuda.com/investig_ESP.html`
3. Garlan, D., Allen, R., Ockerbloom, J. *Achitectural Mismatch or Why it's hard to build systems out of existing parts*. IEEE Software, Noviembre de 1995, págs. 17-26.
4. Dick Hamlet, Dave Mason, and Denise Woit. *Theory of System Reliability Based on Components*. Workshop CBSE - 22rd International Conference on Software Engineering (ICSE'2000). Disponible en línea en `http://www.sei.cmu.edu/cbs/cbse2000/papers/index.html`
5. Itacio: página web del proyecto. `http://www.agustincernuda.com/itacio_ESP.html`
6. Szyperski, Clemens. *Component Software – Beyond Object-Oriented Programming*. Addison-Wesley, 1997.