SciVerse ScienceDirect
Disponible en www.sciencedirect.com

Revista Iberoamericana de Automática e Informática industrial 9 (2012) 290–299

www.elsevier.es/RIAI

Módulo Empresarial para la Validación Formal de Ejercicios aplicado a la Programación Concurrente en Java

P. Basanta-Val*, M. García-Valls, I. Estévez-Ayres y M. J. Martin-Gutiérrez

Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid, Avda. de la Universidad nº 30, 28911, (Leganés) Madrid, España.

Resumen

La utilización de herramientas que permitan detectar problemas de programación es de utilidad tanto para el docente, el cual puede probar de una forma más exhaustiva las prácticas entregadas, como para el discente, el cual puede utilizar dichas herramientas. En muchos casos, existen herramientas previas utilizadas en el desarrollo software, que pueden ser adaptadas para ser utilizadas en un entorno formativo. Este trabajo aporta la integración de una herramienta de validación formal de sistemas concurrentes Java, la cual garantiza la no existencia de defectos como son *el abrazo mortal* y *las condiciones de carrera*, en un entorno Web abierto. Más concretamente, la herramienta que se ha escogido es denominada JPF (Java Path Finder) y se la ha dotado de interfaces dentro de un servidor Java EE (Enterprise Edition), lo que facilita la utilización de servicios propios de la plataforma Java EE y la interoperabilidad entre estos con el módulo diseñado. El artículo trata aspectos tecnológicos derivados de dicha integración como son el diseño de una arquitectura que da soporte a la validación vía web. También detalla una serie de experimentos relativos al rendimiento de la plataforma realizados sobre un curso real, lo que permite medir costes computacionales y su utilidad en la evaluación.

Palabras Clave:
Herramientas, Informática Industrial, Validación Formal, Sistemas Concurrentes, Educación, Java.

1. Introducción

Dentro del currículum de los sistemas embebidos y de tiempo real (Caspi, et al., 2005) (Henzinger, T.A and Sifakis, J., 2007) aparecen los sistemas concurrentes como habilidad necesaria para el discente. Dicha habilidad requiere saber programar sin que se produzcan abrazos mortales (*deadlock*) ni condiciones de carrera (*race conditions*). Los primeros hacen referencia a que varias entidades concurrentes se queden bloqueadas a la espera de un recurso poseyendo otros requeridos por otras entidades concurrentes. Los segundos se refieren a que el proceso de lectura y escritura de una variable de dos o más entidades concurrentes se intercalen en el tiempo.

Comprobar que tanto la condición de carrera como el abrazo mortal no se dan en una aplicación es complejo, utilizándose para tal fin las herramientas de validación formal. Diversas herramientas existentes para lenguajes de programación como C (Volanski, 2008), Ada (Guaspari, Marceau, Polak 1990) o Java (Visser, Pireanu, Khurshid 2004) permiten garantizar formalmente la no existencia de condiciones de carrera ni de abrazo mortal. En esencia dicha comprobación se hace recorriendo todos los estados de un programa verificando si se cumple alguna de invariante que garantice que no hay condición de carrera ni abrazo mortal. Por lo general dicha exploración acaba derivando en una explosión de estados que hace que no sea posible validar totalmente una aplicación.

Aunque estas herramientas tradicionalmente se han utilizado para validar software concurrente, otro tipo de aplicaciones a las que pueden ser adaptadas es el de la enseñanza de la concurrencia propiamente dicha. Así, de la misma forma que un compilador puede dar información sobre si un programa está bien codificado o no, las herramientas de validación formal pueden informar sobre si el programa es seguro (i.e., si potencialmente puede causar condiciones de carrera o/y abrazos mortales) desde un punto de la óptica de un sistema concurrente. Dicha información puede ser utilizada tanto por un docente, en el proceso de evaluación, como por un discente, que pretenda verificar el correcto funcionamiento de su programa.

A la hora de abogar por el uso de técnicas de validación formal en la enseñanza de la concurrencia se debe de tener en cuenta, al menos, dos aspectos importantes. El primero es referente al coste del desarrollo de la validación de las aplicaciones y su complejidad. Por lo general, en entornos académicos la complejidad de los ejemplos desarrollados es menor que en

*Autor de correspondencia a
Correos electrónicos: pbasanta@it.uc3m.es (Pablo Basanta Val),
mvalls@it.uc3m.es (Marisol García Valls),
ayres@it.uc3m.es (Iria Estévez Ayres),
y manu.j.martin@gmail.com (Manuel J. Martín Gutiérrez)
URL: www.it.uc3m.es/drequiem/

doi:10.1016/j.riai.2012.05.013

sistemas reales con el fin de facilitar su comprensión. Eso en principio es una ventaja pues implica que los tiempos de validación presumiblemente serán más bajos en ejercicios académicos que en una aplicación del mundo industrial. El segundo aspecto de importancia es que los ejercicios pueden ser modificados para reducir su complejidad de validación; lo que es más difícil de conseguir en software tradicional atado a requisitos de cliente. De esta manera, muchas de las prácticas pueden ser modificadas para evitar la explosión de estados típica de los validadores formales.

Sin embargo, el coste de la validación formal es mucho más alto que el de compilación. Mientras un programa puede compilar en pocos segundos, es fácil que su validación pueda consumir minutos u horas. Por tanto, la existencia de cierta infraestructura (plataforma) que controle el proceso de validación y optimice el uso de recursos puede resultar de interés. También es de utilidad una arquitectura desacoplada que permita la comunicación asíncrona entre el usuario y la herramienta de validación.

En la plataforma descrita en este artículo, el lenguaje validado es Java y la librería de validación específica utilizada para ese fin es Java Path Finder (JPF) (JPF, 2011) (Visser, Pireanu, Khurshid, 2004). JPF permite la detección de condiciones de carrera y abrazo mortal en aplicaciones Java tradicionales. Está escrita en Java y su código es modificable y adaptable a las necesidades específicas de un escenario concreto como es un curso de programación Java concurrente. En tándem con ella se ha utilizado Java EE (Enterprise Edition) (JavaEE, 2011). Dicha tecnología permite el desarrollo de módulos empresariales (Enterprise Java Beans) que reciben de la infraestructura una serie de servicios genéricos (autenticación, balanceo de carga, nombramiento y gestión de concurrencia) que simplifican el desarrollo de aplicaciones empresariales. Sobre esta tecnología se asienta un módulo que permite validar ejercicios sobre concurrencia en Java.

Aunque la arquitectura descrita ha sido diseñada para la validación de ejercicios en Java (basados en los escenarios descritos en (Estevez-Ayres, Basanta-Val, García-Valls, 2004)), se puede extender para validar otros lenguajes de programación. Puede ser, por ejemplo, aplicado a la enseñanza de programas escritos en C o Ada. Lo que conlleva que los validadores ya existentes (e.g., Vestal (Tomas, Gomez, Perez, 1991)) puedan beneficiarse del modulo desarrollado para gestionar la comunicación con el usuario.

A fin de explicar tanto la arquitectura del sistema de validación como su posterior aplicación, el resto de este artículo se ha estructurado tal y como sigue. La Sección 2 introduce las principales características de la herramienta de validación utilizada (JPF). La Sección 3 describe la estructura del módulo empresarial desarrollado y detalles de su implementación sobre un servidor de aplicaciones. Haciendo uso de una implementación de esta arquitectura, la Sección 4 presenta un estudio de campo asociado a dicha arquitectura en un curso real, con un ejercicio tipo. La Sección 5 conecta el artículo con otras alternativas identificadas en el estado del arte. Finaliza el artículo en la Sección 6 con las conclusiones y líneas futuras derivadas del módulo empresarial implementado.

2. JavaPathfinder

JPF es un software de validación formal para código Java. Básicamente es una máquina virtual de Java (JVM) que ejecuta un programa no sólo una vez, sino teóricamente en todos sus posibles caminos, buscando defectos o errores de ejecución como abrazos mortales o excepciones no manejadas a lo largo de todos los caminos potenciales, además de propiedades que se quieran observar. Cada vez JPF encuentra un fallo, indica la ejecución que ha llevado hasta el mismo. Dicha traza es importante cuando el objetivo es el de arreglar un problema o para comprobar la naturaleza del código generado.

Un inconveniente de JPF es que está implementado en Java, lo que lastra el rendimiento de dicha herramienta. Esto es debido a que una máquina virtual se encuentra ejecutándose sobre otra que se encuentra validando un programa en Java.

Una limitación de JPF es que no puede ejecutar *código nativo*. Esta limitación de JPF impide que las llamadas al sistema para escribir un archivo sean fácilmente deshechas. Con lo cual, JPF no sirve para validar programas escritos en C/C++.

2.1 Modelo de Validación de JPF

El modelo de validación de JPF se basa en un generador de estados específico para Java. Ejecutando instrucciones Java, JPF genera representaciones de estados que pueden ser del siguiente tipo:

- *Comprobado* para igualdad (si el estado ha sido visitado antes).
- *Accedido* para obtener datos (estados de hilos y valores de datos).
- *Almacenado* para ser restaurado más tarde.
- *Restaurado* a partir de una copia almacenada.

Las principales parametrizaciones que admite la JVM de JPF son clases que implementan el manejo de los estados (*matching*, *storing*, *backtracking*). Este esquema de ejecución se asienta sobre la clase `SystemState`, la cual utiliza la clase `SchedulerFactory` (un objeto factoría para `ThreadChoiceGenerators`) para generar secuencias programadas de interés para el validador.

Hay tres métodos de la máquina virtual en el contexto de colaboración máquina virtual–búsqueda (`VM–Search`). La interacción de estos métodos con el paquete de búsqueda es realizada en JPF mediante los siguientes mecanismos:

- `forward` – genera el siguiente estado, informa sobre si el estado generado tiene algún sucesor y si es así, lo almacena en una pila de retorno para una restauración eficiente.
- `backtrack` – restablece el último estado de la pila de retorno.
- `restoreState` – restablece un estado arbitrario (que no necesariamente estuviese en la pila de retorno).

2.2 Mecanismos de Búsqueda en JPF

El objeto búsqueda es el responsable de seleccionar el estado desde el cual la JVM debería proceder, o indicando a la JVM que debe generar el siguiente estado (*forward*), o indicándole que debe regresar a un estado generado anteriormente. Los objetos `Search` se pueden definir como controladores de la máquina virtual. Dichos objetos también configuran y evalúan objetos `property`.

Las principales implementaciones de búsqueda incluyen una simple búsqueda *depth-first* (`DFSearch`) y una búsqueda basada en cola de prioridad, que puede ser parametrizada para hacer diferentes tipos de búsquedas basadas en la selección del estado más interesante del conjunto de todos los sucesores de un estado dado (`HeuristicSearch`). Una implementación simple de

`Search` provee un único método de búsqueda, el cual, incluirá el bucle principal que itera a través del espacio de estados relevantes hasta que ha sido totalmente explorado o la búsqueda encuentra un fallo.

2.3 Algoritmos de Detección de Condición de Carrera y Abrazo Mortal

En (Visser et al. 2005) y (Visser et al. 2003) se proponen una serie de algoritmos para la detección de condiciones de carrera (Eraser) y abrazo mortal (Lock-Tree). Eraser crea una máquina de estados por cada variable compartida y explora la posibilidad de que se escriba una variable por dos hilos al mismo tiempo o uno lea y el otro escriba. Lock-Tree construye un árbol dinámico con los cerrojos que tiene cada hilo del árbol que es analizado más tarde para detectar problemas de abrazo mortal.

Ambos algoritmos son de relevancia para la plataforma diseñada pues son los utilizados a la hora de comprobar el buen funcionamiento de los ejercicios evaluados por la plataforma.

2.4 Optimización POR (Partial Order Reduction)

El número de las diferentes combinaciones planificadas es el principal factor contribuyente a aumentar el tamaño del espacio de estados de los programas concurrentes. En algunos casos prácticos no es necesario explorar todas las posibles intercalaciones de instrucciones para todos los hilos pues el número de estados inducidos planificados puede ser reducido significativamente. De forma práctica, esto se hace agrupando todas las secuencias de instrucciones en un hilo que no puede tener efectos fuera de su ámbito.

JPF emplea un *on-the-fly* POR, una optimización no controlable por el usuario, y que por tanto no está basada en la instrumentación de usuario o en un análisis estático. JPF automáticamente determina, en tiempo de ejecución, qué instrucciones tienen que ser tratadas como fronteras de transición entre estados. De esta manera, si POR está habilitado (configurado mediante las propiedades `vm.por`), una petición avanzada a la JVM y ejecuta todas las instrucciones en el hilo actual hasta que se da una de las siguientes condiciones:

- La siguiente instrucción es relevante para la planificación.
- La siguiente instrucción produce un resultado no determinista.

Desde el punto de vista de la herramienta implementada, la optimización POR es relevante ya que permite reducir los tiempos de validación notablemente. Cuenta con la ventaja de que es una optimización que no necesita configuración previa fuerte y que por tanto es de utilidad cuando se quiere mantener un alto grado de independencia con la aplicación a validar.

Cambiar el mecanismo de búsqueda, mediante nuevos heurísticos, puede ser también bastante efectivo desde el punto de vista temporal, pero tiene una mayor dependencia con el programa a validar. Es por ello que los heurísticos de búsqueda no se exploran en más profundidad en el contexto del módulo empresarial desarrollado.

3. Modelo Desarrollado para el Módulo Empresarial

El modelo desarrollado posee un caso principal que es denominado validación-vía-web (Figura 1). Dicho caso permite tanto al usuario como al administrador de sistema realizar un proceso de validación mediante el uso de la plataforma. Internamente, este caso de uso utiliza cuatro subcasos: `extracción`, `compilación`, `ejecución` y `notificación`. El primer subcaso se encarga de descomprimir los ficheros que llegan al validador (e.g. en un fichero `.zip` o `.jar`). El segundo subcaso de compilar el código fuente de los programas. El tercero se encarga de la ejecución del elemento que realiza la validación formal. Por último, el cuarto se encarga de retornar de forma asíncrona al cliente el resultado de todas estas operaciones (a través del correo electrónico).

Internamente, (Figura 2) el sistema se comunica a través del navegador y del correo electrónico. A través del navegador se envía al servidor la petición al servidor, típicamente encapsulada en un fichero comprimido. Al llegar al servidor, hay un gestor de peticiones que decide si el trabajo se encola a la espera de que haya sitio libre en el sistema para depositarlo, o si por el contrario pasa a validarse. Antes de realizar la validación, son necesarias dos operaciones: una de extracción de ficheros y la otra de compilación. La extracción de ficheros sirve para poder extraer los ficheros en un directorio temporal. Sobre dicho directorio opera el compilador que toma los ficheros `.java` y genera sus correspondientes ficheros `.class`. Tras la compilación pasa a ejecutarse el módulo de validación que comprueba si hay o no problemas de concurrencia.

Por último, los resultados son enviados al cliente. Estos resultados incluirán el informe generado por JPF. En el caso de que la práctica no compile o no pueda descomprimirse, también se envía dicha información al usuario, con lo que siempre se notifica sobre el resultado de la validación al usuario del sistema.

Figura 1. Funcionalidad de validación de ejercicio vía web

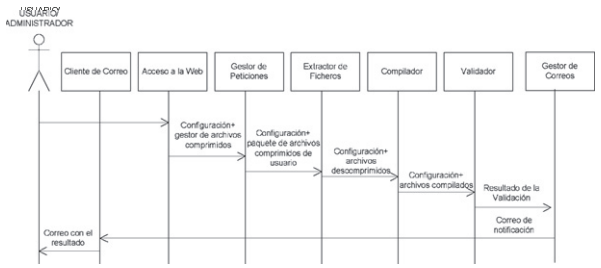

Figura 2. Diagrama de secuencia correspondiente a validación de un ejercicio vía web

3.1 Estructura de Módulos y Relación con Java EE

Este mecanismo de validación forma parte de un sistema de módulos más complejo (Figura 3) donde hay tres clases de módulos:

- *Componentes de Primer Orden*. Los componentes de primer orden atienden al caso de uso general mostrado en la Figura

2. También contienen otros elementos relacionados con cuestiones de acceso a la aplicación y el plano de configuración.

- *Componentes de Segundo Orden*. Este segundo nivel particulariza a nivel tecnológico el primer nivel con aspectos de implementación. Así, algunos componentes del primer nivel son soportados en el segundo nivel con un único elemento; éste es el caso de *validador* que se particulariza en *validar (JPF)*. Otros elementos de primer nivel tienen varios correspondientes de segundo nivel; este es el caso por ejemplo de la extracción de archivos que contempla dos casos: archivo *.zip* y *.jar*. También es el caso de acceso a la aplicación que puede ser web o a través de un cliente de consola.
- *Componentes Java EE*. Este nivel se refiere a los servicios que son provistos por el propio servidor de aplicaciones a los módulos de aplicación. La figura sólo muestra aquellos elementos que son necesarios para que funcione la aplicación desarrollada, a saber:
    - El servicio de acceso web que permite el acceso *http*. Procesa peticiones que llegan del navegador web (su interfaz se presenta en la Figura 11).
    - El contenedor de *EJBs* (JavaEJB, 2011) que permite desplegar la aplicación desarrollada bajo la forma de componente empresarial.
    - El contenedor de *servlets y JSPs* (JavaServ, 2011) que permite colocar lógica de ejecución en el servidor.
    - El servicio JMS (*Java Messaging Service*) (JMS, 2011), el cual, permite el uso de asincronismo en el servidor.
    - El último servicio de Java EE utilizado es el *Java mail* (JMail, 2011) , el cual, permite leer/enviar correo desde código Java.

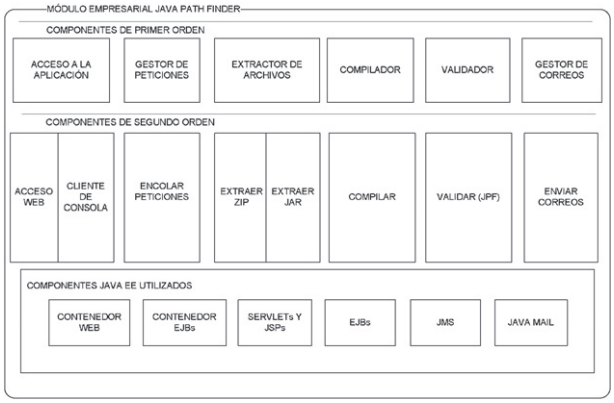

Figura 3. Vista modular del sistema e integración con elementos Java EE

3.2 Módulos de Alto Nivel

Tras detallar los módulos que se implementan, sus componentes y la tecnología utilizada en la integración, se detalla el primer nivel. Este nivel es el que marca el funcionamiento del resto del sistema.

- *Gestor de peticiones*. El módulo de gestión de peticiones está formado por un componente que cumple la función de manejar tanto los datos y los archivos recibidos, como la entrega de estos datos y archivos a los módulos de compilación y validación. Dicho modulo tiene un servlet de recepción. El servlet de recepción maneja los datos recibidos tanto por el cliente de consola como del cliente web. Es el componente encargado de enviar las peticiones recibidas a la cola JMS.
- *Gestor de accesos*. Como la validación es un procedimiento que consume gran cantidad de CPU, se necesita controlar el número de aplicaciones que validan a la vez sin perder las peticiones que se reciban en el caso de que el servidor llegue al límite de validaciones simultáneas. Este componente se dedica a gestionar las peticiones que le llegan y a almacenarlas hasta que puedan ser compiladas y validadas.
La plataforma Java EE permite mantener una cola utilizando un tipo específico de EJB y el EJB de mensajes que utiliza la tecnología JMS.
Este componente, implementado como un EJB de mensajes, obtiene el mensaje enviado por el servlet de recepción vía JMS y lanza una llamada a la compilación y a la validación cuando están disponibles. En la Figura 4 se muestra un esquema básico de funcionamiento de la cola de mensajes.

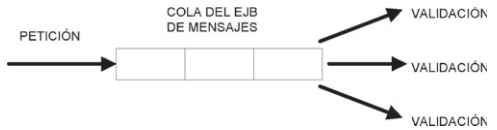

Figura 4. Esquema de funcionamiento de la cola de mensajes JMS

- *Extractor de archivos.* Este módulo recibe del *servlet* de recepción los archivos comprimidos y los descomprime en el directorio indicado. Puede descomprimir tanto archivos *.zip* como con *.jar*. Ambas están implementadas como dos clases Java a las que accede el *servlet* de recepción mediante los métodos que se muestran en la Figura 5 y la Figura 6.

Figura 5. Clase ExtractorZip

Figura 6. Clase ExtractorJar

- *Compilador*. La aplicación JPF necesita tener el código fuente accesible para indicar las líneas de código donde se da el error, por lo que los archivos que se envíen deben ser ficheros `.java`. Sin embargo, para analizarlos utiliza

únicamente ficheros `.class`, por lo que es necesario que haya un componente que compile la aplicación. Este módulo se encarga de gestionar también la jerarquía de carpetas en las que se ubicarán los archivos compilados. Este módulo está implementado como una clase Java normal a la que se accede con los métodos mostrados en la Figura 7.

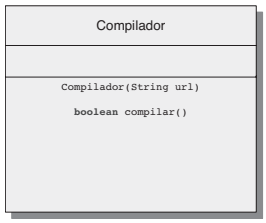

Figura 7. Clase Compilador

- *Validador*. Este módulo se encarga de la validación de la aplicación enviada utilizando para ello la herramienta JPF. Este componente es el núcleo de toda la aplicación. Se limitan el número de aplicaciones que se validan simultáneamente. Para ello se implementa el módulo utilizando la tecnología Java EE y, más concretamente, un EJB de *sesión sin estado* (i.e., que no requiere soporte de la base de datos). Esto permite tener un número de instancias idénticas disponibles que reciben las peticiones del gestor de conexiones. Ese número de instancias es igual al número de aplicaciones que se puede validar simultáneamente y es posible configurarlo en el servidor de aplicaciones.
  El módulo de validación se compone de dos clases:
  - `Ejecutor`: Es una clase Java normal que contiene el código de la llamada a la herramienta JPF con la configuración indicada. El módulo empresarial implementado utiliza el módulo de gestión de correos para enviar el resultado de la validación. Su interfaz se muestra en la Figura 8.
  - `Validador`: Es la clase que implementa el EJB de sesión sin estado. Realiza la llamada al compilador y a la clase `Ejecutor`. Su interfaz se muestra en la Figura 9.

  Una vez que se libera una instancia del EJB de sesión sin estado (`Validador`), el EJB de mensajes del gestor de accesos toma de la cola la siguiente petición y la envía, utilizando para ello la interfaz de acceso.

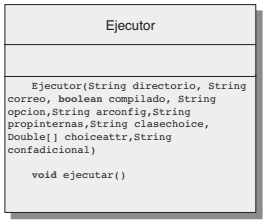

Figura 8. Clase Ejecutor

- *Gestor de correos*. Para informar al usuario de la aplicación sobre el proceso de validación, se utiliza el envío de correos electrónicos, tanto para indicar la finalización del proceso con el resultado de JPF, como para notificar en ciertas partes del proceso indicando el fallo o el funcionamiento correcto del código fuente a validar.
  Este módulo se compone de una clase, llamada `Cartero`, que contiene el código que facilita el envío de correos por parte de otros módulos. Utiliza la librería `javax.mail` de Java y la interfaz que ofrece se muestra en la Figura 10.

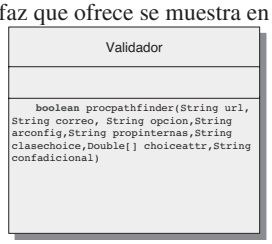

Figura 9. Clase Validador

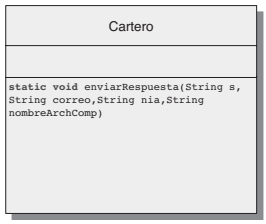

Figura 10. Clase Cartero

Como resultado de los diferentes componentes desarrollados, descritos en las secciones anteriores, se obtienen módulos que cumplen los objetivos marcados. Dichos módulos proveen un acceso web y un acceso vía consola, ambos configurables, que admiten el envío de ficheros comprimidos con ficheros Java dentro. Una vez recibidos los ficheros, se extraen, se compilan y se validan, utilizando como núcleo JPF, para finalmente devolver el resultado al usuario por correo.

3.3 Detalles de Implementación e Interfaz Gráfica

La aplicación ha sido implementada sobre un servidor de código libre *Glassfish* (Glassfish, 2011). Las peticiones, ya sean desde el cliente de consola o desde la página web, las recibe un *servlet*, el cual envía las peticiones a una cola JMS. Un *message driven bean* (MDB) va atendiendo las peticiones de la cola y enviándolas a un Session Bean sin estado, el cual, se encarga de hacer llamadas a clases que controlan la extracción, compilación, ejecución con JPF y del posterior envío del correo con el resultado. El número de aplicaciones simultáneas que atiende el MDB inicialmente se ha configurado a tres, configuración que ofrece buenos rendimientos (mejores que configurada a 4) en un sistema con dos CPUs.

El cliente web puede ser utilizado por todo tipo de usuarios (tanto usuarios normales como administrador), e incluye campos con todo lo necesario para la configuración de la posterior validación. Está compuesto por una página JSP, que incluye la presentación con código HTML, y un *servlet* que gestiona las llamadas a la página y los datos que se envían desde ella. Este *servlet* forma parte de la implementación del módulo de gestión de peticiones.

Los modos que ofrece esta interfaz son:

- *Modo Normal*: Que utiliza la configuración básica de la herramienta JPF, sin ningún tipo de modificadores, tal y como esté configurado por defecto en el servidor.
- *Modo Usuario*: Que utiliza la configuración básica de la herramienta JPF, pero con un archivo de chequeo adicional que el usuario haya incluido en el archivo comprimido enviado.
- *Modo Experto*: Que utiliza una configuración totalmente seleccionada por el usuario. Para esto, se debe rellenar el

formulario que aparece justo debajo de la selección de modo, justo al pinchar en ese modo.

En la Figura 11 se muestra una captura de pantalla correspondiente a dicho cliente web.

Figura 11. Pantalla de acceso web del módulo empresarial

En el Apéndice A se pueden ver el resultado retornado por el módulo a través de correo electrónico. Se incluye tanto en el caso de un fichero que no contiene ningún fallo como en el caso de otro que presenta problemas.

4. Experiencia de Campo con el Módulo Diseñado

Se ha realizado una prueba de su funcionamiento en un curso donde ha sido utilizado como soporte a la evaluación. El resultado de la evaluación es información sobre si un ejercicio es correcto o no, entendiendo por correcto que esté o no libre de fallos (abrazo mortal y/o condición de carrera). Dicha información podría ser utilizada tanto por el docente (para asignar una calificación) como por el discente (para corregir fallos encontrados en un código que desarrolla) si se le ofrece esta oportunidad.

Los objetivos de evaluación han sido:

- Determinar, empíricamente, el coste computacional introducido por el módulo empresarial implementado y estimar si puede estar disponible para el discente. Para ello, se mide de forma empírica el rendimiento de la plataforma y el tiempo que tarda en validar el conjunto de ejercicios (ver Sección 4.2) de una misma entrega.
- Determinar el grado de utilidad del módulo empresarial así como su potencial evaluador. El grado de utilidad se mide viendo el número de prácticas que presentan error en una entrega dada (ver Sección 4.3.1); a mayor número de problemas detectados, mayor potencial del módulo. El potencial evaluador se mide viendo la correlación entre la nota obtenida y la existencia de errores de verificación (ver Sección 4.3.2) de forma estadística.

En ambos casos la experiencia de campo ha sido realizada sobre el mismo escenario, el cual se describe a continuación (Sección 4.1).

4.1 Ejercicio Analizado: Santa Claus

Es una variante de la bien conocida práctica de Santa Claus realizada utilizando Java estándar. Dicha práctica ha sido impartida varios años en ambas asignaturas para enseñar regiones sincronizadas y semáforos.

Esta aplicación se compone de cuatro tipos de hilos y de dos variables compartidas:

- `main`: Es el hilo encargado de inicializar el sistema. Hay un único hilo de este tipo. El funcionamiento de este hilo es tal y como sigue: primero se crea un objeto de la clase `Santa`, al ser un hilo se lanza su ejecución; posteriormente se crean hilos de tipo `Reno` e hilos de tipo `Duende` en distintos bucles y se lanza su ejecución.
- `Santa`: Hay un sólo hilo de este tipo en el sistema. Sus principales funciones son *repartir regalos*, *arreglar problemas y dormir*, en ese orden de preferencias. En un principio el hilo está dormido a la espera de que aparezcan los renos o se hayan producido al menos tres fallos en la cadena de producción. En el primer caso, se simula el reparto de los regalos mediante una espera de 100 ms. y se notifica a los renos su permiso para irse de vacaciones. En el segundo caso, se corrigen todos los fallos que hayan surgido hasta el momento y se notifica a los respectivos duendes para que prosigan su producción. Si se producen ambos casos, primero se reparten los regalos y no se permite a los duendes despertar a `Santa`, pues ya está despierto.
- `Reno`: Sus principales funciones son irse de vacaciones y ayudar a `Santa` a repartir los regalos. Cuando vuelven todos, el último notifica a `Santa` y pasan todos a esperar permiso para irse de vacaciones. Reciben este permiso cuando `Santa` acaba de repartir los regalos. Al llegar cada reno se incrementa una variable compartida por todos y se decrementa cuando se va.
- `Duende`: Su principal función es producir juguetes, aunque en este caso se producen errores temporalmente por lo que deben notificar a `Santa` para que les corrija. Cada hilo empieza simulando el tiempo que tarda en producirse un fallo (2 segundos), incrementa una variable compartida (`contadorDUENDES`) y comprueba que al menos hay tres fallos y que `Santa` está dormido para despertarlo. Se encarga de decrementar la variable compartida. Cada `Duende` espera el permiso de `Santa` para seguir su proceso de producción.

Las variables compartidas que utilizan dichos elementos son:

- `contadorRENOS`: Esta variable es de tipo entero, compartida por los renos (hilos `reno1`, `reno2`, `reno3`). Todos ellos la incrementan cuando llegan de vacaciones a repartir los regalos y la decrementan cuando terminan y se van de nuevo.
- `contadorDUENDES`: Esta variable es de tipo entero, compartido por los duendes (hilos `duende1`, `duende2`, `duende3`) y el hilo `Santa`. Los duendes la incrementan cuando les surge algún error en la producción y `Santa` la `decrementa` cuando corrige el error y lo notifica al duende correspondiente.

En total en el sistema hay 20 Duendes, 9 Renos y 1 Santa interactuando de la forma anteriormente descrita.

Para resolver este problema utilizando el mecanismo de semáforos se suelen utilizar cinco semáforos:

- Tres semáforos inicializados 0. Estos semáforos sirven de barrera a cada uno de las tres entidades de la práctica.
- Un semáforo inicializado a 1. Dicho semáforo sirve para evitar condiciones de carrera sobre una barrera contador.

- Un último semáforo inicializado a 20. Dicho semáforo es utilizado por los duendes.

El módulo empresarial ha sido utilizado para evaluar las prácticas de dos asignaturas impartidas en la Universidad Carlos III de Madrid (ambas comparten la misma práctica):

- Arquitecturas Distribuidas: http://www.it.uc3m.es/labad/
- Sistemas Concurrentes: http://www.it.uc3m.es/tsc/

4.2 Consumo de CPU Derivado del Módulo Empresarial

Se ha analizado el tiempo requerido para hacer validaciones con todas las entregas correspondientes a un curso estándar. En total, el curso constaba de 50 alumnos agrupados en parejas; con lo que hay 25 entregas a validar. Los resultados globales obtenidos están descritos en la Tabla 1. Ésta contiene el tiempo total necesario para validar todas las prácticas (un total de 59 minutos) el escenario del peor caso correspondiente a un escenario donde todas las prácticas tardan lo que tarda la práctica que más tiempo consume (13,5 horas en total para las 25 prácticas); y por último el escenario del mejor caso, donde todas las prácticas tardan lo que tarda la validación más rápida (2 segundos).

Es importante resaltar en estos resultados que los tiempos de validación varían varios órdenes de magnitud. En el caso promedio y el peor hay un orden de magnitud; diferencia que se amplía por encima de los tres órdenes de magnitud cuando se considera el ratio entre el mejor y el peor caso del coste de la validación.

Tabla 1. Resultados para la validación de todo el grupo (Intel Core i7 a 1,60 GHz y 4 GB de memoria RAM)

	Tiempo en horas	Tiempo por práctica (min.)
Caso_Medido	0.98	1.17
Peor_Caso	13.549	32.52
Mejor_Caso	0.007	0.02

Resulta también interesante analizar el coste por práctica (Figura 12) en la escala de segundos y teniendo en cuenta el hecho de si la práctica tenía un error o no. Normalmente, el coste de validación en prácticas con error es menor de lo que es el coste en prácticas donde no hay error (pues hay que explorar más estados).

En los resultados obtenidos se observan tendencias de forma clara:

- El *coste máximo* por práctica es *menor* (un 40%) en prácticas con fallo que en las que no tienen fallo (ver columna MAX en la Figura 12). Este fenómeno se explica por el hecho de que, para detectar que no hay fallo, hay que explorar todos los estados y detectar que no hay ningún problema en todos ellos.
- El *coste promedio* de validación por práctica es mayor en prácticas que no tienen fallo que en las que tienen fallo (el coste se reduce en un 80% en estas últimas). El fenómeno es debido al mismo efecto que en el caso anterior, el coste de detección de fallo es menor que el coste de saber que todo el sistema está bien.
- La *desviación típica* en prácticas sin error es también menor en prácticas sin fallo que en prácticas con fallo. La diferencia es debida a que en las prácticas con fallo éste aparece relativamente de forma temprana.
- El *coste mínimo* en las prácticas libres de error es mucho menor (de 1 segundo) en prácticas que tienen fallo que en las que no (1 minuto).

De forma conjunta, estos resultados muestran que la plataforma puede ser utilizada como complemento para el profesor. El cual puede beneficiarse de ella para realizar el proceso de corrección parcial de las prácticas (esto es, ver si hay o no abrazo mortal o condición de carrera) de forma auxiliada.

También podría ser utilizada por los alumnos para validar sus prácticas enviándolas a través de una página web, recibiendo un correo cuando finaliza la validación de éstas. Sin embargo, los experimentos también nos han mostrado que no se puede hacer un uso fuerte de la plataforma, similar al que se realiza con el compilador (compilación frecuente), con el validador formal, pues las latencias (32 minutos en el peor de los casos) hacen inviable la utilización del módulo empresarial durante el transcurso de la práctica (que dura 2 horas) más de tres o cuatro veces. Su aplicación requiere la utilización de un servidor dedicado accesible desde Internet por los alumnos durante un período de tiempo más largo.

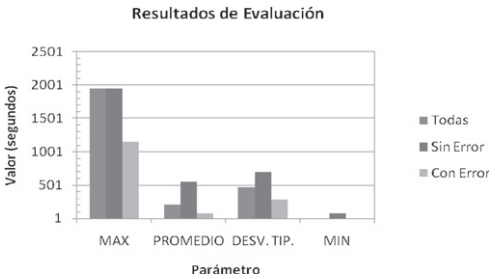

Figura 12. Tiempo consumido por el validador por práctica

4.3 Utilidad del Módulo Empresarial como Elemento de Evaluación y Potencialidad

Desgranados los resultados referentes al rendimiento de la plataforma la evaluación se centra en dos aspectos fundamentales. El primero de ellos es ver la potencialidad del módulo desarrollado y la mejora que puede introducir cuando es aplicado a los ejercicios propuestos. El segundo estudia la correlación entre las notas obtenidas y la información aportada por el módulo empresarial.

Figura 13. Porcentaje de ejercicios con problemas de compilación según el módulo desarrollado

4.3.1 Potencialidad

Para medir el potencial del módulo se han medido el tanto por ciento de prácticas que presentan error. Se entiende como error el que el ejercicio propuesto tenga una condición de carrera o un

abrazo mortal. Los resultados muestran que más de la mitad de las prácticas entregadas presentan problemas de condiciones de carrera o abrazo mortal, según el módulo desarrollado (Figura 13).

Estos resultados son muy relevantes para el módulo desarrollado, pues nos muestra que puede ser muy útil como elemento para mejorar la calidad de las prácticas obtenidas.

De estos resultados parece que puede resultar interesante que se deje interactuar al discente con el módulo empresarial. Más de la mitad de las prácticas podrían beneficiarse de dicha interacción gracias a que se aporta una traza con el error obtenido.

4.3.2 Correlación entre Calificaciones y Estar o no Libre de Errores la Práctica

Al disponerse de las notas obtenidas (calificadas por el docente en absoluto desconocimiento del módulo desarrollado), se ha procedido medir la correlación entre las notas obtenidas por los discentes y el hecho de que el ejercicio tenga o no problemas de concurrencia (salida ofrecida por el módulo empresarial). Los resultados obtenidos (Figura 14) muestran que el promedio y la media son buenos indicadores estadísticos para el conjunto de todas las prácticas. A efectos prácticas la mediana en prácticas que no presentan errores es 0,95 puntos mejor que la mediana obtenida considerando sólo prácticas que presentan error. Si se ve el promedio, la diferencia disminuye a 0,65 puntos.

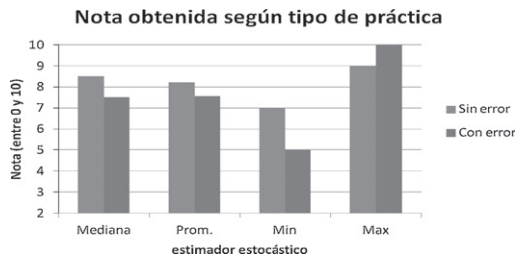

Figura 14. Nota obtenida (mediana, promedio, mínima o máxima) según el tipo de práctica entregada (con error o sin error) de acuerdo al módulo desarrollado

Se han estudiado también el mínimo y el máximo de las notas, pero se ha visto que no son indicadores que correlacionan tan bien con la nota obtenida como la mediana y el promedio. De hecho, en el experimento realizado, el máximo de las prácticas con error supera el máximo de las que no lo tienen (Figura 14).

Preguntados los docentes sobre el esquema de evaluación empleado se ha visto que tiene componentes que tienden a sumar cierta componente no lineal. Esto es debido a que una práctica mal evaluada hace que el discente suspenda. Otra fuente de no linealidad es la memoria sobre la práctica entregada por los alumnos junto a su ejercicio. Teniendo en cuenta todo esto, se ha procedido a ponderar las notas obtenidas, repitiéndose el experimento anteriormente descrito.

Los resultados obtenidos (Figura 15), analizando de nuevo el comportamiento de la mediana y el promedio, muestran que las diferencias entre los ejercicios con error y sin error se hacen más grandes. Así, la diferencia entre la mediana de los ejercicios que presentan error y los que no lo presentan crece desde 0,96 a 2 cuando son reevaluados con la nueva métrica. En el caso del promedio el aumento es mucho mayor (de 0,65 se pasa a 2,56).

En conjunto los resultados obtenidos en esta segunda parte de la evaluación muestran en primer lugar que un gran número de prácticas (64%) se podrían beneficiar del módulo propuesto para eliminar problemas de abrazo mortal y/o condición de carrera de su código. Por otro lado, se ha visto que hay una correlación positiva (en mediana y promedio) entre la calificación existente asignada al alumno y el que la práctica presente o no problemas de concurrencia, información provista por el módulo empresarial.

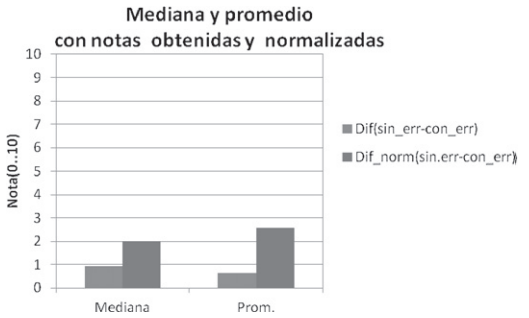

Figura 15. Evolución de la mediana y del promedio cuando las prácticas son reevaluadas

5. Trabajo Relacionado

Java Path Finder (JPF) (Visser, Pireanu, Khurshid, 2004) ha sido utilizado en múltiples proyectos que abarcan desde la validación de programas concurrentes Java, las aplicaciones Web (Rajan, et al., 2009) a un uso específico en las aplicaciones de tiempo real. En este último caso, está siendo utilizado para validar (Kalibera, Parizek, Mlohlava, 2010) aplicaciones basadas en la especificación RTSJ (The Real-Time Specification for Java) (Bollella et al. 2001). El dominio de aplicación estudiado en este proyecto es el de la educación, lo que permite reducir la complejidad (y el tiempo) del proceso de validación.

Dentro del domino educativo, JPF ha sido utilizado para la generación de datos para prácticas (Inhantola, 2006). Inhantola, ha abordado la generación de pruebas, caso distinto del abordado en el trabajo descrito en este artículo. El presente trabajo está más dirigido a la validación de programas concurrentes Java que generalmente tienen condiciones de carrera o abrazos mortales que a la generación de pruebas automáticas. No obstante, ambas aproximaciones se complementan mutuamente. El presente trabajo deja de lado la generación de pruebas automáticas, aunque no impide que éstas se añadan al sistema.

Otra gran diferencia entre ambos usos de la plataforma es el tiempo requerido para realizar la verificación: en nuestro caso es altamente relevante pues hay un actor humano (profesor o alumno) que espera por dicho resultado. Mientras que en (Inhantola, 2006) dichas latencias no son tan importantes pues caen dentro del tiempo de diseño.

Trasladado el contexto al dominio de las aplicaciones codificadas en Ada, destaca la existencia de herramientas como por ejemplo Vestal (Tomas, Gomez, Perez, 1991) y CATS (Guaspari, Marceau, Polak, 1990). La primera de ellas está pensada para la enseñar la programación concurrente en Ada y sus diferentes mecanismos de programación. Tras analizar su descripción, se observa que no posee soporte a la validación formal en temas relacionados con los abrazos mortales o las condiciones de carrera. Por el contrario CATS sí que posee dicho soporte integrado y permite la detección de condiciones de carrera en casos sencillos, en tiempos similares (segundos y minutos) a los ofrecidos por el módulo desarrollado.

La gran diferencia entre el trabajo propuesto en este artículo y CATS está tanto en lenguaje de programación (Java en nuestro caso y Ada en CATS), como en la arquitectura. Mientras la herramienta propuesta es una aplicación web, CATS está al mismo nivel que Java Path Finder: las dos son herramientas de validación genéricas candidatas a ser integradas en otros sistemas. De hecho, la arquitectura descrita en este artículo podría servirle a CATS para poder hacerse accesible remotamente desde navegadores web y/o correo electrónico.

Finalmente, es de destacar que desde el punto de vista de la enseñanza de las aplicaciones concurrentes, siempre ha sido muy relevante la utilización de maquetas pues sirven para visualizar los problemas existentes en un entorno real y experimentar con él. Por ejemplo, (de la Puente, et al., 1998) y (Alonso, Pastor, Alvarez, 2004) presentan un aproximación basada en maquetas (en el primer caso controladas con Ada y en el segundo con Java). Los escenarios en estas dos experiencias pueden ser utilizados como ejercicios que alimenten el módulo empresarial desarrollado, previa virtualización de los recursos físicos descritos en las maquetas.

6. Conclusiones y Trabajo en Curso

En este artículo se ha desarrollado un módulo empresarial para la validación formal de programas concurrentes en Java. El módulo desarrollado posee capacidades de extracción, compilación y validación en entornos fuera de línea donde el usuario envía un fichero comprimido a través de un servidor de aplicaciones al módulo de validación. Dicho módulo devuelve mediante un correo al cliente con el resultado del proceso indicando si ha habido algún problema durante dicho proceso.

Con dicho módulo empresarial se evaluaron las prácticas de un curso donde se imparte programación concurrente en Java. Los tiempos obtenidos nos muestran que la herramienta puede ser beneficiosa a la hora de corregir prácticas y que además puede utilizarse como ayuda para el docente y el discente, bajo ciertas restricciones relativas al coste de la validación formal.

El trabajo futuro se dirige en dos direcciones tecnológicas: la validación de aspectos relacionados con la concurrencia para RTSJ (The Real-Time Specification for Java) y la integración en plataformas educativas como módulo general de otras arquitecturas. En el primer caso, los ejercicios se complementarán con ejemplos académicos de RTSJ (Wellings, 2004). También se pueden incluir ejemplos del *no-heap remote object* tomados de (Basanta-Val, García-Valls, Estevez-Ayres, 2010), (Basanta-Val, García-Valls, Estevez-Ayres, 2005), (Basanta-Val, García-Valls, Estevez-Ayres, 2004) y validar así su regla de asignación formalmente. Como prueba de concepto también se explora validar aspectos de planificación dinámica basada en los conceptos desarrollados en (García-Valls, Alonso, De La Puente, 2012). En el segundo caso, la estrategia prevista pasa por la integración con una arquitectura LMS (Muñoz-Merino, Delgado-Kloos, Fernández-Naranjo, 2009) que ayude al alumno a aprender de forma incremental y organizada. Por último, los autores también evaluarán el impacto que el módulo tiene en el aprendizaje, dejando para ello que el discente use el módulo desarrollado mientras resuelve sus ejercicios.

English Summary

Enterprise Module for Exercise Formal Validation applied on Java Concurrent Programming

Abstract

Tools that allow detecting programming faults are useful for both docents, who may test submitted exercises, and students, who may use these tools in advance. In this article the authors develop one tool for detecting failures in applications. In many cases there are previous tools that may be readapted to be used in an educational scope. This article integrates of one of these tools, which avoids code with dead-locks and race-conditions, into the Internet. The tool integrated is JPF (Java Path Finder) and it is accessed from a Java EE web frontend which carries out the exercise assessment. The article deals with the definition of the module and its evaluation on a realistic scenario. The results show that many assignments may benefit from the output of the tool.

Keywords:

Tools, Industrial Informatics, Formal Validation, Concurrent Systems, Education, Java

Agradecimientos

Este trabajo ha sido realizado con el apoyo del proyecto iLAND (ARTEMIS-JU 100026) parcialmente financiado por ARTEMIS JTU y el Ministerio de Industria, Comercio y Turismo español. También ha sido parcialmente financiado por ARTISTDesign NoE (IST-2007-214373) del 7º programa marco de la Unión Europea, por REM4VSS (TIN2011-28339) del Ministerio de Ciencia e Innovación y por LEARN3(TIN2008-0513) del Ministerio de Ciencia e Innovación. Los autores quieren también agradecer a sus anónimos revisores las mejoras sugeridas en las versiones previas de este artículo. Sus comentarios, realizados como si fuesen coautores más del artículo, han servido para hacer que este artículo sea más claro y conciso. También han ayudado a identificar vías para continuar el trabajo presente en futuros artículos.

Referencias

Alonso D., Pastor, J.A., Alvarez, B., 2004. Real-Time Teaching with Java: JPR 3. En: OTM Workshops. Larnaca (Chipre)

Basanta-Val, P., Garcia-Valls, M. y Estevez-Ayres, I. , 2010. No-Heap remote objects for distributed real-time Java. ACM Trans.Embed.Comput.Syst. 10(1), 1-25.

Basanta-Val, P., Garcia-Valls, M. y Estevez-Ayres, I., 2005. Towards the Integration of Scoped Memory in Distributed Real-Time Java. ISORC 2005. Seatle(US)

Basanta-Val, P., Garcia-Valls, M. y Estevez-Ayres, I., 2004. No Heap remote objects: Leaving the garbage collector at the server-side. En: OTM Workshops. Larnaca (Chipre)

Bollella G., et al., 2001. The Real-Time Specification for Java, Adisson-Wesley.

Caspi P., Sangiovanni-Vincentelli, A.L, Almeida, L., Benveniste, A., Bouyssounouse, B., Buttazzo, G. C., Crnkovic, I., Damm, W., Engblom, J., Fohler, G., García-Valls, M., Kopetz, H., Lakhnech, Y., Laroussinie, F., Lavagno, L., Lipari, G., Maraninchi, F., Peti, P., de la Puente, J.A, Scaife, N., Sifakis, J., de Simone, R., Törngren, M., Veríssimo, P., Wellings, A. J., Wilhelm, R. Willemse, Wang Yi, T.A.C., 2005. Guidelines for a graduate curriculum on embedded software and systems. ACM Trans. Embedded Comput. Syst. 4(3), 587-611

de La Puente, J., Alonso, A. , Garcia-Valls, M. , Ruiz, J.F., 1998. Teaching real-time systems at DIT/UPM," En: Real-Time Systems, Montreal (Canada).
de Tomas, M. A., Gomez, L., Perez A., 1991. Vestal: a tool for teaching concurrency in Ada. En: Proceedings of the conference on TRI-Ada '91: today's accomplishments; tomorrow's expectations. USA.
Estévez-Ayres, I., Basanta-Val, P., García-Valls, M., 2004. Docencia de Programación Concurrente. Experiencias de laboratorio. En: VII Jornadas de Tiempo Real. Málaga, Spain.
García-Valls, M., Alonso, A., De La Puente, J.A., 2012. A dual-band priority assignment algorithm for dynamic QoS resource management. Accepted in Future Generation Computer Systems. doi:10.1016/j.future.2011.10.005
Glassfish, Servidor GlassFish, disponible en octubre del 2011 desde http://glassfish.java.net
Guaspari, D., Marceau, C., Polak, W., 1990. Formal Verification of Ada Programs. IEEE Transactions on Software Engineering 16(9), 1058-1075
Henzinger, T.A, Sifakis, J. (2007) The Discipline of Embedded Systems Design. IEEE Computer 40(10): 32-40.
Ihantola, P., 2006. Test data generation for programming exercises with symbolic execution in Java PathFinder. En: 6º Baltic Sea conference on Computing education research. USA.
JavaEJB, Enterprise Java Beans Container, disponible en octubre del 2011 desde http http://jcp.org/en/jsr/detail?id=220
JavaEE, Java Enterprise Edittion, disponible en octubre del 2011 desde oracle.com/technetwork/java/javaee/
JPF. Java Path Finder, disponible en octubre del 2011 desde http://javapathfinder.sourceforge.net
JavaServ, Java Servlets, disponible en octubre del 2011 desde http://jcp.org/en/jsr/detail?id=340
JMail, Java Mail, disponible en octubre del 2011 desde http://jcp.org/en/jsr/detail?id=919
JMS, Java Messaging System, disponible en octubre del 2011 desde http://jcp.org/en/jsr/detail?id=914
Kalibera, T. Parizek, P., Malohlava, M., 2010. Exhaustive Testing of Safety Critical Java. En: JTRES'10, 2010 Prague,Czech Republic
Muñoz-Merino, P.J., Delgado-Kloos, C., Fernández-Naranjo, J., 2009. Enabling interoperability for LMS educational services. Computer Standards & Interfaces 31(2), 484-498
Rajan, S.P, Tkuchuk, O., Prasad, M., Ghosh, I., Goel, N., 2009. WEAVE: Web Applications Validation Environment. En: ICSE'09. Vancouver (Canada).
Visser, W., Pireanu, C.S., Khurshid, S., 2004. Test input generation with java PathFinder. SIGSOFT Softw. Eng. Notes 29(4), 97-107.
Visser,W., Havelund, K., Brat, G., Park,S., Lerda,. F. Model Checking Programs. Automated Software Engineering Journal. Volume 10, Number 2, April 2003.
Volanschi, N., 2008.A portable compiler-integrated approach to permanent checking. Journal: Automated Software Engineering 15(1) 21-37
Wellings, A., 2004. Concurrent and Real-Time Programming in Java. Wiley.

Apéndice A

Este apéndice contiene el listado resultado de la validación de la práctica de Santa Claus. El Listado 1 se corresponde con la práctica de Santa (Santa.java) descrita en la parte experimental. Esta práctica compila no presenta errores y valida en aproximadamente un minuto aproximadamente. El Listado 2 muestra un ejemplo donde la aplicación tiene un problema de abrazo mortal.

```
JavaPathfinder - UC3M v5.0 - (C) RIACS/NASA Ames Research Center
====================================================== system
under test
application: Santa.java

====================================================== search
started: 12/01/11 11:43

====================================================== search
constraint
Free Memory Limit: 1048576

====================================================== snapshot
thread index=1,name=Thread-
0,status=WAITING,this=Reno@325,priority=5,lockCount=0,suspendCount=0
 waiting on: java.lang.Thread$Permit@342
 call stack:
(…)
thread index=30,name='Thread-
29,status=RUNNING,this=Nicolas@475,priority=5,lockCount=0,suspendCount=0
 call stack:
      at java.util.concurrent.Semaphore.release(Semaphore.java:-1)
      at java.util.concurrent.Semaphore.release(Semaphore.java:60)
      at Nicolas.run(Santa.java:242)

====================================================== results
no errors detected

====================================================== statistics
elapsed time:       0:00:48
states:             new=2930, visited=0, backtracked=0, end=0
search:             maxDepth=2930, constraints=1
choice generators:  thread=2930, data=0
heap:               gc=2963, new=3398, free=2917
instructions:       68427
max memory:         989MB
loaded code:        classes=82, methods=1301

====================================================== search
finished: 12/01/11 11:44
```

Listado 1. Resultado cuando la aplicación no presenta errores

```
JavaPathfinder - UC3M v5.0 - (C) RIACS/NASA Ames Research Center
======================================================
system under test
application: Santa.java

====================================================== search
started: 12/01/11 11:43

====================================================== error #1
gov.nasa.jpf.jvm.NotDeadlockedProperty
deadlock encountered:
 thread
index=0,name=main,status=TERMINATED,this=null,target=null,priority=5,lockCount=
0,suspendCount=0
 thread index=1,name=Thread-

(…)
thread index=31,name=Thread-
30,status=WAITING,this=Duende@744,priority=5,lockCount=0,suspendCount=0
 waiting on: java.lang.Thread$Permit@755
 call stack:
      at java.util.concurrent.Semaphore.nativeAcquire(Semaphore.java:-1)
      at java.util.concurrent.Semaphore.acquire(Semaphore.java:44)
      at Duende.run(Santa.java:220)

====================================================== choice trace
#1

====================================================== results
error #1: gov.nasa.jpf.jvm.NotDeadlockedProperty "deadlock encountered:    thread
index=0,name=main,..."

====================================================== statistics
elapsed time:       0:00:09
states:             new=1520, visited=134, backtracked=134, end=1
search:             maxDepth=1520, constraints=0
choice generators:  thread=1520, data=0
heap:               gc=1716, new=1462, free=836
instructions:       25905
max memory:         123MB
loaded code:        classes=87, methods=1322

====================================================== Search
finished: 12/01/11 11:43
```

Listado 2. Resultado cuando la aplicación presenta un abrazo mortal (deadlock)