Ingeniería Investigación y Tecnología, volumen XV (número 4), octubre-diciembre 2014: 493-504
ISSN 1405-7743 FI-UNAM
(artículo arbitrado)

Pruebas empíricas y resultados preliminares con la herramienta Krakatoa para Full Static Program Verification

Empirical Tests and Preliminary Results with the Krakatoa Tool for Full Static Program Verification

Ramírez-de León Edgar Darío
División Académica de Informática y Sistemas
Universidad Juárez Autónoma de Tabasco
Correo: msc-dar_ram@hotmail.com

Chávez-Bosquez Oscar
División Académica de Informática y Sistemas
Universidad Juárez Autónoma de Tabasco
Correo: chavez@programador.com

Francisco-León Julián Javier
División Académica de Informática y Sistemas
Universidad Juárez Autónoma de Tabasco
Correo: juljav_fl@hotmail.com

Información del artículo: recibido: mayo de 2012, reevaluado: junio de 2013, aceptado: septiembre de 2013

Resumen

XJML (Ramírez *et al.*, 2012) es una plataforma modular externa para la verificación y validación de clases Java empleando el lenguaje de modelado de Java (JML, *Java Modeling Language*) a través de contratos escritos en XML. Uno de los problemas que se enfrentaron durante el desarrollo de XJML fue la integración de la técnica de verificación denominada *Full Static Program Verification* (FSPV). En este artículo se presentan los experimentos y resultados que permitieron definir qué herramienta se integra en XJML para ejecutar FSPV.

Descriptores:

- verificación de modelos
- verificación y validación
- JML
- XML
- Java
- *Runtime Assertion Checking*
- *Extended Static Checking*
- *Full Static Program Verification*

Abstract

XJML (Ramírez et al., 2012) is a modular external platform for Verification and Validation of Java classes using the Java Modeling Language (JML) through contracts written in XML. One problem faced in the XJML development was how to integrate Full Static Program Verification (FSPV). This paper presents the experiments and results that allowed us to define what tool to embed in XJML to execute FSPV.

Keywords:

- *model checking*
- *verification and validation*
- *JML*
- *XML*
- *Java*
- *Runtime Assertion Checking*
- *Extended Static Checking*
- *Full Static Program Verification*

Introducción

JML fue creado por Gary T. Leavens y es el *Behavioral Interface Specification Language* (BISL) más popular para Java (Chalin *et al*., 2008; Chalin *et al*., 2007), soportando *Runtime Assertion Checking* (RAC), *Extended Static Checking* (ESC) y *Full Static Program Verification* (FSPV), de hecho, es el único BISL soportado por estas tres tecnologías de verificación (Chalin *et al*., 2008; Chalin *et al*., 2007).

Desde la concepción de JML se han efectuado múltiples avances en el desarrollo de herramientas para su soporte. Esto nos conduce a una gran cantidad de herramientas, duplicación de trabajo (esfuerzos) y una alta y colectiva sobrecarga para el mantenimiento de JML y sus herramientas de soporte. Sin embargo, la evolución de Java como lenguaje de programación, y las mejoras aplicadas a las herramientas y aplicaciones de JML, obliga a la comunidad de JML a mantenerse al día, que es una tarea difícil en donde se producen muchas variantes y alternativas de dichas herramientas y aplicaciones.

Estas razones y el hecho de que aún no existe una herramienta que brinde soporte total a RAC, ESC y FSPV tal como se presenta en Ramírez *et al*. (2012) fue lo que impulsó el diseño y desarrollo de XJML. XJML es, hasta donde sabemos, la primera propuesta para embeber un contrato para una clase Java usando JML y XML. XJML 1.0 logra integrar:

a) RAC empleando como base JML4 (Chalin *et al*., 2008), sobre todo los componentes `jml4c` y `jml4rt`
b) ESC a través de ESC/Java2 (Flanagan *et al*., 2002), versión 2.0.5

Si bien la integración de estas herramientas soportadas a través de XML representó un reto interesante, fue aún mayor el problema que se encontró al integrar FSPV en XJML 1.0. Dicho esto, el objetivo de este trabajo es presentar los experimentos y resultados que permitieron determinar la herramienta que da soporte a FSPV en XJML 1.0, `FSPVRunner` como se denomina a dicho componente en XJML 1.0 (Ramírez *et al*., 2012).

Marco referencial

Como se ha indicado en la sección anterior, JML es un lenguaje para la verificación de clases Java que soporta las técnicas de Verificación de Aserciones en Tiempo de Ejecución (RAC, *Runtime Assertion Checking*), Verificación Estática Extendida (ESC, *Extended Static Checking*) y la Verificación de Programas Completamente Estática (FSPV, *Full Static Program Verification*). Pero, ¿qué son estas técnicas de verificación, cuál es su función u objetivo primordial?

RAC consiste en la verificación en tiempo de ejecución de las aserciones expresadas en algún lenguaje de especificación (para el caso de este trabajo, en JML). Las aserciones son una de las técnicas automatizadas más usadas para la detección de fallos en sistemas computacionales a la vez que proveen información sobre su localización, inclusive para fallas (*faults*) que se producen durante la ejecución, pero no conducen a errores significativos (*failures*).

Las aserciones tienen una larga y distinguida historia en los anales de la ingeniería de *software* y el diseño de lenguajes de programación. Inicialmente se desarrollaron como un medio para expresar las propiedades deseadas de un programa como un paso necesario en la construcción de pruebas formales y deductivas para la corrección de programas.

Las aserciones han tenido muchas otras aplicaciones en la ingeniería de *software* a través de los años (Clarke y Rosenblum, 2006), aunque primordialmente en las etapas posteriores de desarrollo, en particular en el desarrollo y ejecución del código fuente. Son un elemento importante de la verificación de modelos (*model checking*), una técnica alternativa y activamente estudiada de la verificación de programas (*program verification*), en el cual el espacio de estados resultante de la ejecución de un programa se verifica contra aserciones lógicas que expresan propiedades de seguridad temporal (*temporal safety*) y de tiempo de vida o vitalidad de un programa (*liveness*) (Clarke y Rosenblum, 2006).

ESC es la verificación estática de las aserciones expresadas en JML. Es una técnica experimental de verificación de programas en tiempo de compilación (Flanagan *et al*., 2002). Una de las herramientas más conocidas para esta verificación en Java es ESC/Java, la cual emplea un generador de condiciones de verificación (*verification-condition generation*) y técnicas de demostración automática de teoremas (Flanagan *et al*., 2002). Provee a los programadores, a través de un lenguaje simple de anotaciones, decisiones de diseño que pueden expresarse formalmente.

ESC/Java examina el programa anotado y notifica inconsistencias entre las decisiones de diseño expresadas en las anotaciones y el código actual (implementación) y también advierte sobre errores potenciales en tiempo de ejecución del código.

FSPV es una técnica de verificación que emplea demostradores de teoremas (*theorem provers*) (Karabotsos *et al*., 2008). Esta técnica de verificación de programas puede emplear demostradores automáticos de

teoremas, como por ejemplo[1]: Alt-ergo, Simplify, Z3, Yices, CVC3, CVCL, Gappa, Eprover, haRVey, Zenon, SPASS, Vampire y veriT, o bien, emplear asistentes o demostradores interactivos (requieren la intervención humana) de teoremas, tales como[1] Coq, PVS, Isabelle/HOL, HOL 4, HOL Light y Mizar.

La demostración automática de teoremas (ATP, *automated theorem proving*) que también puede llamarse deducción automatizada, es actualmente el sub-campo más desarrollado del razonamiento automático, y se encarga de la demostración de teoremas matemáticos mediante programas de ordenador.

Así, un demostrador automático de teoremas (*automatic theorem prover*) es un programa de cómputo desarrollado para la deducción automática de teoremas, ampliamente estudiado y utilizado en la verificación y validación de sistemas de cómputo.

Por otro lado, un asistente de pruebas (*proof assistant*) o demostrador interactivo de teoremas (*interactive theorem prover*) es una herramienta de *software* para asistir en el desarrollo de demostraciones formales mediante la colaboración hombre-máquina. Esto involucra algún tipo de editor interactivo de demostraciones, o alguna otra interfaz, con la cual un humano puede guiar la búsqueda de demostraciones.

Se preguntará: ¿qué tiene que ver todo esto con XJML o incluso con este trabajo?, ¿qué es Krakatoa, qué relación y cuál es su importancia en este trabajo?

XJML 1.0 adopta la plataforma Why para efectuar FSPV, ya que esto es posible gracias al complemento (*plug-in*) Krakatoa que forma parte de la plataforma Why. Krakatoa es una herramienta de FSPV para clases Java anotadas con JML (Karabotsos *et al*., 2008) y es similar a FSPV Theory Generator (FSPV TG) (James y Chalin, 2009, Chalin *et al*., 2008), (Karabotsos *et al*., 2008), en el sentido que traduce programas Java a un programa intermedio. Sin embargo, los programas en Why se traducen en una teoría específica usando el compilador Why escrito en Objective CAML.

XJML 1.0 soporta Why 2.30 y Why3. Why 2.30 es una plataforma para la verificación de *software* que contiene varias herramientas[2]:

- Why, un Generador de Verificación de Condición (VCG, *Verification Condition Generator*) de propósito general, el cual se utiliza como herramienta de fondo (*back-end*) por otras herramientas de verificación (Krakatoa y Caduceus) pero también puede usarse de manera directa para la verificación programas.
- Krakatoa[3], una herramienta para la verificación de programas Java.
- Caduceus[4], herramientas para la verificación de programas hechos en el lenguaje C; sin embargo Caduceus es ahora obsoleto y en su lugar, los usuarios deben cambiar a Frama-C[5].

Por otra parte, Why3[6] es la siguiente generación de la plataforma Why para la verificación de software. Separa claramente la parte puramente lógica de la especificación y la generación de verificación de condiciones de los programas.

Why3 se desarrolla en el proyecto de equipo ProVal[7], el cual está conformado por INRIA Saclay-Île-de-France, en conjunto con el Laboratorio de Investigación en Informática de París (Laboratoire De Recherche En Informatique) y la Universidad del Sur de París (University of Paris Sud).

En este trabajo se presenta un conjunto de experimentos que ayudaron a determinar la herramienta que da soporte a FSPV en XJML 1.0, `FSPVRunner` como se denomina a dicho componente en XJML 1.0 (Ramírez *et al*., 2012) y que pueden considerarse como una extensión dentro del proyecto XJML.

Para concluir con esta sección resta describir, de manera breve, qué es `FSPVRunner`. `FSPVRunner` es un componente del subsistema `Runners` de XJML, que se encarga de ejecutar FSPV a través de la herramienta Krakatoa de la plataforma Why. La plataforma XJML es descrita con mayor detalle en Ramírez *et al.* (2012).

El nombre `FSPVRunner` se conforma por el prefijo FSPV, el cual corresponde a dicha técnica de verificación, y el sufijo *Runner* se debe a que `FSPVRunner` es un componente que forma parte del subsistema lanzadores (*Runners*) de XJML 1.0. Dicho esto, `FSPVRunner` se encarga de ejecutar FSPV y su arquitectura y funcionamiento es descrito con mayor detalle en Ramírez *et al.* (2012).

Método

Todos los experimentos se llevaron a cabo en un equipo de cómputo con las siguientes características en *hardware* y *software*:

- Notebook Dell Vostro 1520
- Procesador Intel(R) Core(TM) 2 Duo CPU T6670 a 2.20 Ghz

1 En la sección de resultados y discusión encontrará la URL de descarga de cada uno de ellos.
2 http://why.lri.fr/
3 http://krakatoa.lri.fr/
4 http://why.lri.fr/caduceus/
5 http://frama-c.cea.fr/
6 http://why3.lri.fr/
7 http://proval.lri.fr/

- Memoria RAM de 4 GB
- Sistema Operativo GNU/Linux Ubuntu 10.04 (lucid) de 32 bits, núcleo Linux 2.6.32-34- generic con GNOME 2.30.2
- Java Development Kit 6.0 update 29
- Eclipse SDK versión 3.7.0, build id I20110613-1736

Se otorgaron 30 segundos para la ejecución de cada experimento y el proceso para determinar la herramienta que da soporte a FSPV en XJML 1.0 consistió, primero en probar los módulos FSPV para JML4, JMLEclipse y OpenJML, los cuales son los últimos esfuerzos de la comunidad de JML para desarrollar un entorno integrado de verificación (integrated verification environment) (Chalin *et al*., 2010). Respecto a JML4 FSPV TG Theory Generator (James y Chalin, 2009; Chalin *et al*., 2008; Karabotsos *et al*., 2008), se descartó debido a su inestabilidad observada en las pruebas al momento de instalar y configurar los módulos requeridos[8].

Las conclusiones sobre la inestabilidad y etapa de desarrollo de JML4 y JMLEclipse están fundamentadas en las páginas Web principales de los respectivos proyectos[9]. Puede observarse cómo en dichas páginas se menciona que el lanzamiento de una primera versión oficial (no beta) está pendiente por determinarse.

Otra herramienta con la que se efectuaron pruebas fue OpenJML. OpenJML es la siguiente generación del núcleo de JML y será soportado por Java 1.7. Está basada en OpenJDK, por lo que desafortunadamente la instalación fallará (Cok, 2011) si la VM de Java no es una versión compatible con Java 1.7. Además, aún hay mucho trabajo por hacer respecto a OpenJML y sus características como un *plug-in* de Eclipse aún no están definidas (Cok, 2011). En resumen, OpenJML tampoco está listo para utilizarse de manera estable.

Como ya se mencionó, integrar RAC y ESC en XJML 1.0 no presentó mayor dificultad en comparación con FSPV, tal como hemos presentado. Ahora bien, si JML4, JMLEclipse y OpenJML no pudieron proveer un módulo de FSPV estable, fue necesario buscar alguna herramienta o plataforma que pudiera soportar tal técnica de verificación en XJML 1.0. En la tabla 1 se presenta una comparación de herramientas para FSPV, lo que aunado con los argumentos presentados, permitieron determinar la plataforma Why (Christophe y Marché, 2007) como herramienta de soporte de FSPV para XJML 1.0.

Tabla 1. Comparación de herramientas FSPV para Java (Karabotsos *et al.,* 2008)

	LOOP	JACK	Krakatoa Why	FSPV TG Simpl
Maintained	×	×	✓	✓
Open Source	×	✓	✓	✓
Proven Sound	✓	×	✓	✓[1]
Proven Complete	×	×	×	✓[1]
Above two proofs done	in PVS	N/A	by hand	in Isabelle
VC generation done in prover	×	×	×	✓
Termination of recursive functions	×	×	×[2]	✓

[1] Simpl is proven sound and complete. The translation to Simpl is not.

Para su mejor comprensión se describe a continuación cada leyenda de las filas de la tabla 1.

- *Maintained:* indica si a la herramienta aún se le da soporte o se mantiene en desarrollo.
- *Open source:* indica si la herramienta es de código abierto.
- *Proven sound*: indica si se ha demostrado que la herramienta es consistente[10].
- *Proven complete:* indica si se ha demostrado que la herramienta es completa (nos referimos a la completitud en el campo de lógica computacional).
- *Above two proofs done:* indica en qué herramienta, técnica o método fue demostrada la consistencia y completitud de la herramienta.
- *VC generation done in prover:* indica si la herramienta es capaz de generar sus propias condiciones de verificación (VC es acrónimo de *verification-condition*).
- *Termination of recursive functions:* indica si la herramienta tiene la capacidad de tratar con funciones recursivas.

Con la determinación de la plataforma Why para dar soporte a FSPV en XJML 1.0, fue necesario delimitar qué versión de la plataforma Why soportaría XJML 1.0, así como cuáles demostradores automáticos de teoremas (*Automatic Theorem Provers*) se utilizarían para verificar el comportamiento de FSPV en XJML 1.0.

8 Tal como se especifica en http://sourceforge.net/apps/trac/jmlspecs/wiki/JML4%20Setup y http://sourceforge.net/apps/trac/jmlspecs/wiki/JML4%20HowTo%20Run%20Tests.
9 http://sourceforge.net/apps/trac/jmlspecs/wiki/JML4 y http://sourceforge.net/apps/trac/jmlspecs/wiki/JmlEclipse.

10 Las propiedades de solidez o consistencia (*soundness*) y completitud (*completness*) son las que se piden a cualquier método de validación. La propiedad de solidez nos garantiza que el método solo nos dará conclusiones válidas y la completitud nos garantiza que podremos obtener todas las conclusiones derivables.

Se efectuaron pruebas con Why 2.30 (Christophe, 2003) y Why3 0.71 (Bobot *et al.*, 2011) que son las versiones disponibles hasta el momento de la redacción de este trabajo. Para cada versión de la plataforma Why se realizaron experimentos con sus respectivos demostradores automáticos de teoremas soportados, de acuerdo con la documentación disponible para cada versión de Why (Christophe, 2011).

Los experimentos consistieron en verificar la clase `Isqrt.java` (figura 1), la cual es una implementación que calcula la raíz cuadrada de un número entero.

La implementación de la clase `Isqrt` es breve, pero a la vez creemos que fue lo suficientemente completa para emitir la conclusión que permitió seleccionar las herramientas que dieron soporte a FSPV en XJML 1.0. Así, la clase `Isqrt` contiene al menos una precondición (línea 10 de la figura 1), postcondición (línea 11, figura 1), invariante de clase (líneas 2 a 8 en la figura 1) y de ciclo (líneas 16 a 18, figura 1), las cuales son los tipos de aserciones más comunes en la verificación y validación de sistemas computacionales.

Será necesario presentar los distintos demostradores automáticos de teoremas y demostradores interactivos de teoremas (*proof assistant* o *interactive theorem prover*) que se usaron para realizar los experimentos con la clase `Isqrt`, mismos experimentos que permitieron seleccionar qué versión de la plataforma Why, demostradores automáticos de teoremas y demostradores interactivos de teoremas emplear para efectuar los experimentos con el módulo FSPV de XJML 1.0. Se efectuaron experimentos con Why 2.30 y Why3 0.71 con los demostradores automáticos de teoremas:

- Alt-ergo 0.93
- Simplify 1.5.4
- Z3 2.2 y 3.2
- Yices 1.0.17 y 1.0.31
- CVC3 versión development y 2.4.1
- CVCL versión development y 2.4.1
- Gappa 0.12.0 y 0.15.1
- Eprover 1.4 Namring
- haRVey 0.5.10
- Zenon 0.6.3
- SPASS 3.7
- Vampire 0.6
- veriT 201107

Se efectuaron experimentos con Why 2.30 y Why3 0.71 empleando los siguientes demostradores interactivos de teoremas

- Coq 7.4 y 8.3pl2
- PVS 5.0
- Isabelle/HOL 2011-1
- HOL 4 (Kananaskis 7)
- HOL Light 100110
- Mizar 7.12.01

Al igual que con los demostradores automáticos de teoremas listados, la lista de los demostradores interactivos de teoremas se obtuvo de la documentación y páginas Web de Why 2.30 y Why3 0.71, por lo que es posible que existan más de estos, con los cuales no fue posible efectuar experimentos debido a que no se encuentran en la lista de los soportados por Why.

```
1.  //@+ CheckArithOverflow = no
2.  /*@ lemma distr_right:
3.    @   \forall integer x y z; x*(y+z) == (x*y)+(x*z);
4.    @*/
5.  /*@ lemma distr_left:
6.    @   \forall integer x y z; (x+y)*z == (x*z)+(y*z);
7.    @*/
8.  //@ logic integer sqr(integer x) = x * x;
9.  public class Isqrt {
10.     /*@ requires x >= 0;
11.     @ ensures
12.     @      \result >=0 && sqr(\result) <= x
13.     @      && x < sqr(\result + 1); @*/
14.     public static int isqrt(int x) {
15.         int count = 0, sum = 1;
16.         /*@ loop_invariant
17.           @   count >= 0 && x >= sqr(count) && sum == sqr(count+1);
18.           @ loop_variant x - sum; @*/
19.         while (sum <= x) {
20.           count++; sum = sum + 2*count+1;
21.         }
22.         return count;
23.     }
24. }
```

Figura 1. Clase de prueba para determinar las herramientas que dieron soporte a FSPV en XJML 1.0

Resultados y discusión

Las tablas 2 y 3 muestran la URL de descarga de los demostradores automáticos de teoremas y demostradores interactivos de teoremas empleados en los experimentos.

Ahora bien, las tablas 4 y 5 muestran, respectivamente, los distintos demostradores automáticos de teoremas y demostradores interactivos de teoremas utilizados para efectuar experimentos iniciales con el fin de observar el comportamiento de Why 2.30 y Why3 0.71 con cada uno de ellos y posteriormente experimentar con la clase `Isqrt`.

Las columnas de las tablas 4 y 5 se describen a continuación:

- Plataforma Why. Versión de la plataforma Why en la que se utilizó el demostrador automático de teoremas o el demostrador interactivo de teoremas.
- Versión. Versión del demostrador automático de teoremas o demostrador interactivo de teoremas.
- Versión recomendada. De acuerdo con la documentación de la plataforma Why, se indica la versión que se recomienda para el demostrador automático de teoremas o demostrador interactivo de teoremas.
- Compilado. Indica si el demostrador automático de teoremas o demostrador interactivo de teoremas fue compilado en la misma máquina donde se realizaron los experimentos.
- Binario. Si el demostrador automático de teoremas o demostrador interactivo de teoremas no fue compilado en la misma máquina donde se realizaron los experimentos, entonces, se utilizó una versión precompilada del mismo. Se trata de una versión compilada en otro equipo, distinto al que fue utilizado para realizar los experimentos.
- Información mostrada. Se incluye la salida presentada por cada versión de la plataforma Why al detectar los demostradores automáticos de teoremas o demostradores interactivos de teoremas soportados.

De acuerdo con los resultados presentados en las tablas 4 y 5 se seleccionaron los demostradores automáticos de teoremas y demostradores interactivos de teoremas donde el valor de la columna **Información mostrada** sea **Ok** o **None** (sin ninguna nota informativa o de advertencia). La tabla 6 muestra la codificación utilizada para tabular los resultados de las tablas 7 y 8, las cuales presentan los demostradores interactivos de teoremas y demostradores automáticos de teoremas, respectivamente, que se emplearon para probar la clase `Isqrt.java`. Estos experimentos permitieron seleccionar con qué versión de la plataforma de Why, con cuáles demostradores automáticos de teoremas y cuáles demostradores interactivos de teoremas se realizaron los experimentos para verificar el módulo `FSPVRunner` de XJML 1.0.

Conclusiones

Los experimentos y resultados presentados pueden servir como base para investigadores que deseen obtener

Tabla 2. URL de descarga de los demostradores automáticos de teoremas empleados en los experimentos

Demostrador automático de teoremas	Versión	URL de descarga
Alt-ergo	0.93	http://alt-ergo.lri.fr/http/alt-ergo-0.93.tar.gz
Simplify	1.5.4	http://www.kindsoftware.com/products/opensource/archives/Simplify-1.5.5-13-06-07-binary.zip
Z3	2.2	http://why.lri.fr/z3-2.2.tar.gz
	3.2	http://research.microsoft.com/projects/z3/z3-3.2.tar.gz
Yices	1.0.17	http://yices.csl.sri.com/cgi-bin/yices-license.cgi?file=yices-1.0.17-i686-pc-linux-gnu.tar.gz
	1.0.3	http://yices.csl.sri.com/cgi-bin/yices-newlicense.cgi?file=yices-1.0.31-i686-pc-linux-gnu-static-gmp.tar.gz
CVC3	devel	http://www.cs.nyu.edu/acsys/cvc3/download/1.5/linux32/cvc3-optimized
	2.4.1	http://www.cs.nyu.edu/acsys/cvc3/releases/2.4.1/cvc3-2.4.1.tar.gz
CVCL	devel	http://www.cs.nyu.edu/acsys/cvc3/download/1.5/linux32/cvc3-optimized
	2.4.1	http://www.cs.nyu.edu/acsys/cvc3/releases/2.4.1/cvc3-2.4.1.tar.gz
Gappa	0.12.0	https://lipforge.ens-lyon.fr/frs/download.php/159/gappa-0.12.0.tar.gz
	0.15.1	https://gforge.inria.fr/frs/download.php/29004/gappa-0.15.1.tar.gz
Eprover	1.4 Namring	http://www4.in.tum.de/~schulz/WORK/E_DOWNLOAD/V_1.4/E.tgz
haRVey	0.5.10	http://harvey.loria.fr/harvey-0.5.10.tgz
Zenon	0.6.3	http://zenon-prover.org/zenon-0.6.3.tar.gz
SPASS	3.7	http://www.spass-prover.org/download/sources/spass37.tgz
Vampire	0.6	http://www.vprover.org/download.cgi
veriT	201107	http://www.verit-solver.org/distrib/veriT-201107.tar.gz

información sobre el comportamiento de la plataforma Why 2.30 y Why3 0.71, con sus respectivos demostradores automáticos de teoremas y demostradores interactivos de teoremas. Por otro lado, y sin olvidar el objetivo que da origen a este trabajo, los resultados permitieron determinar qué versión o versiones de la plataforma Why soportaría XJML 1.0.

Finalmente, la versión de la plataforma de Why así como los demostradores automáticos de teoremas y demostradores interactivos de teoremas fueron seleccionados de acuerdo con los criterios mostrados en la tabla 9.

Debido a la gran discrepancia de resultados entre las dos versiones de la plataforma Why con distintos demostradores automáticos de teoremas (tablas 6 y 7), se decidió realizar los experimentos con ambas versiones de la plataforma Why empleando únicamente aquellos demostradores automáticos de teoremas soportados por ambas versiones lo que permitió emitir conclusiones sobre dichas discrepancias en los resultados.

Dicho esto, XJML 1.0 soporta, a través de `FSPVRunner`, las siguientes versiones de la plataforma Why: Why 2.30 y Why3 0.71.

Actualmente se trabaja con experimentos, empleando la clase `CuentaBancaria` (Oriat, 2005). La clase será verificada en XJML 1.0 empleando las técnicas de verificación: RAC, ESC y FSPV. Después de esto, nos aseguraremos que no existan alteraciones entre los resultados presentados por XJML 1.0 y los resultados de cada técnica de verificación, ejecutadas por separado.

Cabe destacar que los resultados de este trabajo, permitieron delimitar los demostradores de teoremas

Tabla 3. URL de descarga de los demostradores interactivos de teoremas empleados en los experimentos

Demostrador interactivo de teoremas	Versión	URL de descarga
Coq	7.4	ftp://ftp.inria.fr/INRIA/Projects/LogiCal/coq/V7.4/coq-7.4-Linux-i386.tar.gz
	8.3pl2	http://coq.inria.fr/distrib/V8.3pl2/files/coq-8.3pl2.tar.gz
PVS	5.0	http://pvs.csl.sri.com/cgi-bin/downloadlic.cgi?file=pvs-5.0-ix86-Linux-allegro.tgz
Isabelle/HOL	2011-1	http://www.cl.cam.ac.uk/research/hvg/isabelle/dist/Isabelle2011-1_bundle_x86-linux.tar.gz
HOL 4	HOL 4	http://sourceforge.net/projects/hol/files/hol/kananaskis-7/kananaskis-7.tar.gz/download (Kananaskis 7)
HOL Light	100110	http://www.cl.cam.ac.uk/~jrh13/hol-light/hol_light_100110.tgz
Mizar	7.12.01	ftp://mizar.uwb.edu.pl/pub/system/i386-linux/mizar-7.12.01_4.166.1132-i386-linux.tar

Tabla 4. Versiones de la plataforma Why y sus respectivas salidas al detectar distintos demostradores automáticos de teoremas

Plataforma Why	Demostrador automático de teoremas	Versión	Versión Recomendada	Compilado	Binario	Información mostrada
Why 2.30	Alt-ergo	0.93[1]	0.8	Sí	No	None
Why3 0.71						Ok
Why 2.30	Simplify	1.5.4[2]	1.5.4	No	Sí	None
Why3 0.71						Ok
Why 2.30	Z3	2.2[3]	1.3[4]	No	Sí	(obsolete)
Why3 0.71						is quite old…
Why 2.30		3.2[5]		No	Sí	(not supported)
Why3 0.71						is not known to be supported…
Why 2.30	Yices	1.0.17	1.0.17	No	Sí	(obsolete)
Why3 0.71						
Why 2.30		1.0.31[6]	1.0.17	No	Sí	(not supported)
Why3 0.71						is not known to be supported…
Why 2.30	CVC3	devel[7]	1.5	No	Sí	(not supported)
Why3 0.71						is not known to be supported, use it at your own risk!
Why 2.30		2.4.1[8]		Sí	No	(not supported)
Why3 0.71						is not known to be supported…

1 A la fecha del 20 de noviembre de 2011 no encontramos en el sitio versión previa para realizar pruebas.
2 A la fecha del 20 de noviembre de 2011 es la última versión disponible y estable.
3 Versión ofrecida por los autores en el sitio de Why platform.
4 A la fecha del 20 de noviembre de 2011 no encontramos la versión 1.3 en el sitio http://research.microsoft.com/en-us/um/redmond/projects/z3/older_z3.html. Existen paquetes MSI para las versiones 1.3.3, 1.3.5 y 1.3.6, más sin embargo no fueron probadas debido al sistema operativo.
5 A la fecha del 20 de noviembre de 2011 es la última versión disponible y estable.
6 A la fecha del 20 de noviembre de 2011 es la última versión disponible y estable.
7 No logramos compilar la versión development (1.5) debido a errores en el proceso, por ello, descargamos el binario pre-compilado en http://www.cs.nyu.edu/acsys/cvc3/download/1.5/linux32/cvc3-optimized.
8 Para compilar el código fuente se necesitan satisfacerse las siguientes dependencias: GNU/Flex, GNU/Bison, GNU/GMP. Más información en http://cs.nyu.edu/acsys/cvc3/doc/INSTALL.html.

Tabla 4. Versiones de la plataforma Why y sus respectivas salidas al detectar distintos demostradores automáticos de teoremas (... continuación)

Plataforma Why	Demostrador automático de teoremas	Versión	Versión recomendada	Compilado	Binario	Información mostrada
Why 2.30 Why3 0.71	CVCL	devel[1]	1.5	No	Sí	(not supported)
Why 2.30 Why3 0.71		2.4.1[2]		Sí	No	(not supported) is not known to be supported…
Why 2.30 Why3 0.71	Gappa	0.12.0	0.12.0	Sí[3]	No	(obsolete) is quite old, please consider upgrading to a newer version
Why 2.30 Why3 0.71		0.15.1[4]				none ok
Why 2.30 Why3 0.71	Eprover	1.4 Namring	No se especifica	Sí	No	none[5] ok
Why 2.30/ Why3 0.71	haRVey[6]	0.5.10	No se especifica	No	No	none
Why 2.30/ Why3 0.71	Zenon[7]	0.6.3	No se especifica	Sí	No	none
Why 2.30 Why3 0.71	SPASS	3.7	No se especifica	Sí	No	none[8] ok
Why 2.30 Why3 0.71	Vampire	0.6	No se especifica	No	Sí[9]	none ok
Why 2.30 Why3 0.71	veriT[10]	201107	No se especifica	Sí	No	undetected is not known to be supported…

1 No logramos compilar la versión development (1.5) debido a errores en el proceso, por ello, descargamos el binario pre-compilado en http://www.cs.nyu.edu/acsys/cvc3/download/1.5/linux32/cvc3-optimized.
2 Para compilar el código fuente se necesitan satisfacer las siguientes dependencias: GNU/Flex, GNU/Bison, GNU/GMP. Más información en http://cs.nyu.edu/acsys/cvc3/doc/INSTALL.html.
3 Para compilar el código fuente se necesitan satisfacer las siguientes dependencias: GNU/MPFR (instalado desde repositorio, con nombre libmpfr-dev, versión 2.4.2-3ubuntu1) y bibliotecas de Boost C++ (instalado desde repositorio, con nombre libboost1.40-dev, versión 1.40.0-4ubuntu4). Más información en http://lipforge.ens-lyon.fr/www/gappa/doc/index.html.
4 A la fecha del 20 de noviembre de 2011 es la última versión disponible y estable.
5 Posiblemente no soportado por Why2.30 platform.
6 Proyecto aún en etapa de desarrollo tal como lo especifica su página principal http://harvey.loria.fr/.
7 Proyecto aún en etapa de desarrollo tal como lo especifica su página principal http://zenon-prover.org/.
8 No soportado por Why2.x, solo soportado por Why3 tal como se especifica en http://krakatoa.lri.fr/.
9 A través del sitio solo puede descargar archivos binarios para distintas plataformas debido al acuerdo de licencia especificado en http://www.vprover.org/license.cgi.
10 Es el sucesor de haRVey y se encuentra en arduo desarrollo como lo especifica su página principal http://www.verit-solver.org/.

Tabla 5. Versiones de la plataforma Why y sus respectivas salidas al detectar distintos demostradores interactivos de teoremas

Plataforma Why	Demostrador interactivo de teoremas	Versión	Versión recomendada	Compilado	Binario	Información mostrada
Why 2.30 Why3 0.71	Coq	7.4[1]	Al menos 7.4	No	Sí	(obsolete) is not known to be supported, use it at your own risk!
Why 2.30 Why3 0.71		8.3pl[2]		Sí	No	(not supported) Ok
Why 2.30 Why3 0.71	PVS	5.0	No especifica	Sí	No	(not supported) Ok
Why 2.30 Why3 0.71				No	Sí	(not supported) None20
Why 2.30/ Why3 0.71	Isabelle/HOL	2011-1*	No especifica	No	No	None
Why 2.30/ Why3 0.71	HOL 4	HOL 4 (Kananaskis 7)*	No especifica	No	No	None
Why2.30/ Why3 0.71	HOL Light	100110*	No especifica	No	No	None
Why 2.30/ Why3 0.71	Mizar	7.12.01*	No especifica	Sí	No	None

1 No logramos compilar la versión debido al error:

```
Camlp4: Uncaught exception: DynLoader.Error ("pa_ifdef.cmo", "file not found in path")
File "lib/pp.ml4", line 1, characters 0-1:
Error: Preprocessor error
make: *** [lib/pp.cmo] Error 2.
```

2 Posiblemente no soportado por Why3.0.71 platform.
* Proceso de compilación con error(es) y/o integración con Why no fue posible.

(tanto automáticos como interactivos) que serán utilizados para los experimentos con la clase `CuentaBancaria`. De esta manera, los experimentos con la clase `CuentaBancaria` serán realizados con Alt-ergo 0.93, Simplify 1.5.4 y Gappa 0.15.1. No obstante, `FSPVRunner` no limita los demostradores automáticos de teoremas y/o demostradores interactivos de teoremas que puede emplear Why, más bien esta característica es determinada por el soporte que cada versión de la plataforma Why ofrezca.

Trabajos relacionados

El *lenguaje de modelado de java* (JML) es quizás el lenguaje mejor conocido para el modelado de clases Java. Ha evolucionado a través de los años, pasando de JML2, JML3, JML4, JML5, JMLEclipse, y OpenJML, el cual es, hasta donde sabemos, el último esfuerzo de la comunidad de JML para producir un Entorno Integrado de Verificación.

Respecto a JMLEclipse, el primer lanzamiento (*release*) oficial aún está por determinarse[11]. Por otra parte, OpenJML, que es la siguiente generación de herramientas del núcleo de JML y soporta Java 1.7, está basado en OpenJDK y desafortunadamente la instalación falla si la máquina virtual de Java (JVM) no es la adecuada versión 1.7[12]. Por ello, existe mucho trabajo por hacer, y las características u opciones de la interfaz gráfica de

11 Vampire 0.6 no siempre arroja los mismos resultados
12 Como se indica en http//sourceforge.net/apps/trac/jmlspecs/wiki/JmlEclipse

Tabla 6. Codificación para tabular los resultados de los experimentos con Why, demostradores automáticos de teoremas y demostradores interactivos de teoremas

Codificación	Representa
*	A la fecha del 20 de Noviembre de 2011 es la última versión disponible y estable
N/S	No soportado
?	Indeterminado, ya que entre múltiples ejecuciones los resultados no siempre fueron los mismos
CME	Consistente entre múltiples ejecuciones (3 ejecuciones)
NTV/NTT	Número de teorías válidas/Número de teorías totales
NTI/NTT	Número de teorías inválidas/número de teorías totales
NTETEV	Número de teorías que excedieron el tiempo establecido de validación
NTNVOR	Número de teorías que no pudieron validarse por otras razones
N/E	No editada

Tabla 7. Demostradores interactivos de teoremas seleccionados para probar la clase Isqrt.java

Plataforma Why	Demostrador interactivo de teoremas	Versión	CME	NTV/NTT	NTI/NTT	NTETEV	NTNVOR
Why 2.30	Coq	8.3pl2*	N/S	N/S	N/S	N/S	N/S
Why3 0.71			Sí	N/E	N/E	N/E	N/E

Tabla 8. Demostradores automáticos de teoremas seleccionados para probar la clase Isqrt.java

Plataforma Why	Demostrador automático de teoremas	Versión	CME	NTV/NTT	NTI/NTT	NTETEV	NTNVOR
Why 2.30	Alt-ergo	0.93*	Sí	6/6	0/6	0/6	0/6
Why3 0.71			Sí	5/6	0/6	1/6	0/6
Why 2.30	Simplify	1.5.4*	Sí	4/6	0/6	0/6	2/6
Why3 0.71			Sí	0/6	2/6	4/6	0/6
Why 2.30	Gappa	0.15.1*	Sí	0/6	6/6	0/6	0/6
Why3 0.71			Sí	2/6	4/6	0/6	0/6
Why 2.30	Eprover	1.4 Namring*	N/S	N/S	N/S	N/S	N/S
Why3 0.71			Sí	4/6	0/6	2/6	0/6
Why 2.30	haRVey2[1]	0.5.10*	N/S	N/S	N/S	N/S	N/S
Why3 0.71			N/S	N/S	N/S	N/S	N/S
Why 2.30	Zenon2[2]	0.6.3*	N/S	N/S	N/S	N/S	N/S
Why3 0.71			N/S	N/S	N/S	N/S	N/S
Why 2.30	SPASS	3.7*	N/S	N/S	N/S	N/S	N/S
Why3 0.71			Sí	3/6	0/6	3/6	0/6
Why 2.30	Vampire	0.6*	Sí	4/6	0/6	2/6	0/6
Why3 0.71			No	?/6	?/6	?/6	?/6

1 Proyecto aún en etapa de desarrollo tal como lo especifica su página principal http://harvey.loria.fr/
2 Proyecto aún en etapa de desarrollo tal como lo especifica su página principal http://zenon-prover.org/

Tabla 9. Criterios para seleccionar la plataforma de Why que dará soporte a `FSPVRunner`

Criterio	Why 2.30	Why3 0.71
Consistencia de resultados	Sí	No[1]
Soporte continuo de la plataforma	No	Sí
Soporte para múltiples demostradores automáticos de teoremas y demostradores interactivos de teoremas	No	Sí
Los resultados son estables	Sí	No

1 Como se indica en http://sourceforge.net/apps/trac/jmlspecs/wiki/JmlEclipse

usuario (GUI) del complemento (*plug-in*) de Eclipse aún no están definidas[13].

XJML emplea el Lenguaje de Marcas Extensible (XML, *Extensible Markup Language*) como lenguaje para escribir el contrato para una clase Java. El uso de XML se debe a su importante papel en el intercambio de una gran variedad de datos. En este sentido, PiXL (*Protocol Interchange using XML Languages*) (Del-Mar *et al.*, 2007) es un claro ejemplo de cómo las tecnologías y estándares basados en XML pueden utilizarse para la integración de herramientas de análisis. Sin embargo, PiXL no tiene soporte para JML.

Existe una comunidad de JML experimentando con el lenguaje, y los últimos esfuerzos se centran en construir un Entorno Integrado de Verificación (Chalin *et al.*, 2008), el cual soporte e integre RAC, ESC y FSPV.

De estas tres técnicas de verificación, FSPV parece ser la más difícil de integrar y el último intento por lograr esto (OpenJML) aún no está listo. Creemos que XJML es el primero en su tipo, donde las expresiones del lenguaje pueden ser embebidas en un archivo XML (el contrato) para una clase Java, en lugar de la misma clase. Además, hasta donde sabemos XJML es la primera herramienta que integra RA, ESC y FSPV, y es este último el componente de interés en este trabajo, y que a diferencia de JML4 y JMLEclipse, XJML emplea la plataforma Why gracias al componente `FSPVRunner` del subsistema `Runners` de XJML.

En este último aspecto, creemos que este es el primer trabajo donde se realizan experimentos y se analizan los resultados del componente `FSPVRunner` de XJML y que puede ser el primero de una serie de experimentos para verificar la eficiencia y eficacia de XJML, con el fin de identificar puntos débiles o áreas de trabajo que apoyen en la verificación de programas Java.

13 Tomado de The OpenJML User Guide, página 13, disponible en http://jmlspecs.sourceforge.net/OpenJMLUserGuide.pdf

Trabajos futuros

Actualmente se trabaja en experimentos con la clase `CuentaBancaria` y el componente `FSPVRunner`. Después de esto se analizarán y compararán los resultados asegurando que no existan alteraciones entre los resultados y la ejecución de FSPV a través de la plataforma Why.

Respecto a XJML 1.0 cabe destacar que soporta precondiciones, postcondiciones e invariantes de clase, así que se espera que en las siguientes actualizaciones de XJML podamos ofrecer soporte para más expresiones de JML. La tercera cosa por hacer es construir una Interfaz Gráfica de Usuario para XJML.

Respecto a `FSPVRunner` y las pruebas presentadas en este trabajo, se ejecutarán los experimentos en al menos un equipo más, con más de 4 GB de RAM, un microprocesador con más de dos núcleos y velocidad mayor de 2.20 GHz.

Para los experimentos deberá asignarse un tiempo mayor a 30 segundos, que fue el tiempo asignado en este trabajo, por lo que se recomienda se ejecuten primordialmente en Why3 0.71 con Alt-ergo 0.93, debido a que fueron los de mayor éxito en FSPV.

Agradecimientos

Este trabajo fue parcialmente patrocinado por el proyecto: Desarrollo de una infraestructura tecnológica como apoyo a la Comisión de Derechos Humanos del Estado de Tabasco, usando herramientas de *software* de vanguardia en atención a grupos vulnerables, en atención a un proyecto de Fondos Mixtos y en atención a la convocatoria del CONACYT 2008. Clave del proyecto: TAB–2008–C03–95740.

Queremos agradecer también a Gary T. Leavens, Kuat Yessenov, Greg Dennis, y Homero Alpuín por todo su apoyo, respuestas de correo electrónico, sugerencias y más.

Referencias

Bobot-François, Christophe-Filliâtre J., Marché-Claude, Paskevich-Andrei. Why3: Shepherd your Herd of Provers, Boogie 2011: First International Workshop on Intermediate Verification Languages (1er, 2011, Wroclaw, Poland). BOOGIE 2011, first International Workshop on Intermediate Verification Languages, Wroclaw, Poland, Microsoft Research, 2011, pp. 53-64 [en línea]. Disponible en: http://research.microsoft.com/en-us/um/people/moskal/boogie2011/boogie_2011_all.pdf

Chalin P., Chalin P.R., James P.R., Lee J., Karabotsos G. Towards an Industrial Grade IVE for Java and Next Generation Research Platform for JML. *International Journal on Software Tools for Technology Transfer (STTT)* volumen 12 (número 6), noviembre de 2010 [fecha de consulta: 06 de mayo 2012] [en línea]. Disponible en: http://formal.korea.ac.kr/~jlee/papers/sttt10.pdf

Chalin P., James P.R., Karabotsos G. VSTTE '08 Proceedings of the 2nd International Conference on Verified Software: Theories, Tools, Experiments, USA, Springer Berlin–Heidelberg, 2008, cap. 9, JML4: Towards an Industrial Grade IVE for Java and Next Generation Research Platform for JML [fecha de consulta: 06 de mayo 2012] [en línea]. Disponible en: http://www.springerlink.com/content/t28606827773492v/fulltext.pdf

Chalin P., James P.R., Karabotsos. Proceedings of the 2007 Conference on Specification and Verification of Component-Based Systems: 6th Joint Meeting of the European Conference on Software Engineering and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, New York, NY, USA, Association for Computing Machinery, 2007, cap. 6, An Integrated Verification Environment for JML: Architecture and Early Results [fecha de consulta: 06 de mayo 2012] [en línea]. Disponible en: http://users.encs.concordia.ca/~chalin/papers/2007-SAVCBS-Chalin-James-Karabotsos-IVE-for-Jml.pdf

Christophe-Filliâtre J. *The WHY Verification Tool*. Tutorial and Reference Manual, France, INRIA Saclay & Laboratoire de Recherche en Informatique, 2011, 40 p. [en línea]. Disponible en: http://why.lri.fr/manual/manual.ps

Christophe-Filliâtre J. y Marché C. Proceedings of the 19th International Conference on Computer Aided Verification, USA, Springer Berlin-Heidelberg, 2007, cap. 21, The Why/Krakatoa/Caduceus Platform for Deductive Program Verification [fecha de consulta. 06 de mayo 2012] [en línea]. Disponible en: http://www.lri.fr/~filliatr/ftp/publis/cav07.pdf

Christophe-Filliâtre J. Why: a Multi-Language Multi-Prover Verification Tool, Paris, Laboratoire de Recherche en Informatique, Université Paris Sud, 2003, Research Report 1366.

Clarke L.A. y Rosenblum D.S. A Historical Perspective on Runtime Assertion Checking in Software Development. *ACM SIGSOFT Software Engineering Notes*, volumen 31 (número 3), mayo de 2006 [fecha de consulta: 06 de mayo 2012] [en línea]. Disponible en: http://dl.acm.org/citation.cfm?id=1127900&CFID=62138933&CFTOKEN=88416605

Cok David R. *The OpenJML User Guide*, USA, jmlspecs, 2011, 20 p. [en línea]. Disponible en: http://jmlspecs.sourceforge.net/OpenJMLUserGuide.pdf

Del-Mar-Gallardo M., Martínez J., Merino P., Nuñez P., Pimentel E. PiXL: Applying XML Standards to Support the Integration of Analysis Tools for Protocols. *Science of Computer Programming Journal,* volumen 65 (número 1), marzo de 2007 [fecha de consulta: 06 de mayo 2012] [en línea]. Disponible en: http://www.lcc.uma.es/pedro/publications/pixl_gallardo.pdf

Flanagan C., Leino K.R., Lillibridge M., Nelson G., Saxe J.B., Stata R. *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation*, New York, NY, USA, Association for Computing Machinery, 2002, cap. 21, Extended Static Checking for Java [fecha de consulta: 06 de mayo 2012] [en línea]. Disponible en: http://research.microsoft.com/en-us/um/people/leino/papers/krml103.pdf

James P. R. y Chalin P. Proceedings of the 2009 ACM Symposium on Applied Computing, New York, NY, USA, Association for Computing Machinery, 2009, cap. 9, Extended Static Checking in JML4: Benefits of Multiple Prover Support [fecha de consulta: 06 de mayo 2012] [en línea].Disponible en: http://users.encs.concordia.ca/~chalin/papers/JamesChalin09-ACM-SAC-SVT09-preprint.pdf

Karabotsos G., Chalin P., James P.R. y Giannas L. Total Correctness of Recursive Functions using JML4 FSPV, Seventh International Workshop on Specification and Verification of Component-Based Systems (7o, 2008, Atlanta, Georgia, USA). International Workshop on Specification and Verification of Component-Based Systems, Atlanta, Georgia, USA, School of Electrical Engineering and Computer Science, University of Central Florida, 2008, pp.19-26 [en línea]. Disponible en: http://www.eecs.ucf.edu/SAVCBS/2008/papers/Karabotsos-etal.pdf

Oriat C. Jartege. A Tool for Random Generation of Unit Tests for Java Classes, en Quality of Sofware Architectures and Software Quality, 2nd International Workshop of Software Quality (2o, 2005, Erfurt, Germany). Quality of Sofware Architectures and Software Quality, 2nd International Workshop of Software Quality, Erfurt, Germany, LNCS 3712, 2005, pp. 242-256 [en línea]. Disponible en: http://membres-liglab.imag.fr/oriat/soqua.pdf

Ramírez-de-L. E.D., Chávez-Bosquez O., Francisco-L. J.J. Verification and Validation for Java Classes Using Design by Contract. The Modular External Approach. *Revista World Academy of Science, Engineering and Technology* (WASET), volumen 64 (número 14), abril de 2012 [fecha de consulta: 18 de abril de 2012] [en línea]. Disponible en: http://www.waset.org/journals/waset/v64/v64-14.pdf

Este artículo se cita:

Citación estilo Chicago

Ramírez-de León, Edgar Darío, Oscar Chávez-Bosquez, Julián Javier Francisco-León. Pruebas empíricas y resultados preliminares con la herramienta Krakatoa para Full Static Program Verification. *Ingeniería Investigación y Tecnología*, XV, 04 (2014): 493-504.

Citación estilo ISO 690

Ramírez-de León E.D., Chávez-Bosquez O., Francisco-León J.J. Pruebas empíricas y resultados preliminares con la herramienta Krakatoa para Full Static Program Verification. *Ingeniería Investigación y Tecnología*, volumen XV (número 4), octubre-diciembre 2014: 493-504.

Semblanzas de los autores

Edgar Darío Ramírez-de León. Maestro en sistemas computacionales y licenciado en sistemas computacionales por la División Académica de Informática y Sistemas (DAIS) de la Universidad Juárez Autónoma de Tabasco (UJAT). Profesor-Investigador en la DAIS-UJAT. Miembro del Sistema Estatal de Investigadores (SEI) del estado de Tabasco. Ingeniero de *Software*. Áreas de interés: verificación y validación de sistemas, *model checking*, inteligencia de negocios, *data warehouse* y *data mining*.

Oscar Chávez-Bosquez. Profesor-investigador de la división académica de informática y sistemas de la Universidad Juárez Autónoma de Tabasco. Maestro en sistemas computacionales, ha escrito diversos artículos tratando temas modelos formales de desarrollo de *software*, y de optimización combinatoria en el área de metaheurísticas aplicadas a problemas NP-Completos. Ha participado en proyectos con financiamiento externo. Actualmente cuenta con el perfil PROMEP y es miembro del Sistema Estatal de Investigadores.

Julián Javier Francisco-León. Desarrollo y transferencia de tecnología en sectores productivos con necesidades en el área de multimedia. Integran soluciones tecnológicas utilizando elementos multimedios, animación así como sistemas inmersivos 3D y realidad virtual. Ha participado en proyectos desarrollando sistemas para los museos Parque Museo de La Venta y Museo Regional de Antropología Carlos Pellicer Cámara.

Nota. Los tres autores participaron, entre otros, en un proyecto de Fondos Mixtos financiado por CONACYT-UJAT-CEDH titulado: "Desarrollo de una infraestructura tecnológica como apoyo a la comisión de Derechos Humanos del Estado de Tabasco usando herramientas de *software* de vanguardia en atención a grupos vulnerables", con Julián Javier Francisco-León como director.