RACCIS, 2(2), 44-51, 2012.

Revista Antioqueña de las
Ciencias Computacionales y la Ingeniería de Software

ISSN: 2248-7441

www.fundacioniai.org/raccis
raccis@fundacioniai.org

Formal Methods in Industry

Los Métodos Formales en la Industria

Alexei Serna A.
Instituto Antioqueño de Investigación. *alexei.serna(At).fundacioniai.org*

INFORMACIÓN DEL ARTÍCULO

Tipo de artículo: Revisión.

Historia del artículo
Recibido: 15-06-2012
Correcciones: 28-09-2012
Aceptado: 3-10-2012

Categories and Subject Descriptors
D.2.4 [**Software Engineering**]: Software/Program Verification – *Formal Methods.*

General Terms
Software Engineering, Verification, Formal Methods.

Keywords
Formal methods, formalization, Verification, notation.

Palabras clave
Métodos formales, Verificación, formalización, notación.

ABSTRACT
The application of formal methods in industry has progressed extensively over the past decade and the results are promising. But despite these achievements and it have been documented in numerous studies, it is still very common the skepticism about its usefulness and applicability. The goal of this paper is to show that its evolution over the past decade exceeds all previous processes and each time they do a better job to satisfy industrial needs. This is achieved by the description of some experiments and the result of various applications in industry and through an analyzing of the needs of companies that must be satisfy the research community in this field.

RESUMEN
La aplicación de los métodos formales en la industria ha progresado ampliamente en la última década y los resultados son prometedores. Pero, a pesar de estos logros y de que se han documentado en numerosos estudios, todavía es muy común el escepticismo acerca de su utilidad y aplicabilidad. El objetivo de este artículo es mostrar que su evolución en la última década sobrepasa todos los procesos anteriores y que cada vez hacen un mejor trabajo para satisfacer las necesidades industriales. Esto se logra mediante la descripción de algunos experimentos y resultados de diversas aplicaciones documentadas en la industria y mediante un análisis a las necesidades de las empresas, que debe satisfacer la comunidad investigadora en este campo.

1. INTRODUCCIÓN

La industria del software ha utilizado y documentado los métodos formales en varios estudios de caso [1, 2], pero aunque los logros han sido positivos, el escepticismo acerca de su utilidad todavía es muy común. Las quejas más frecuentes se refieren a que no se adaptan a los problemas industriales, que son demasiado costosos, o que exigen conocimientos y formación más allá de lo que se encuentra disponible en el mercado laboral. Aun así, diversos investigadores los exploran y experimentan desde hace varios años y los aplican en casos reales de la industria [2-4]. Aunque no todos los experimentos han tenido éxito, hoy más que en cualquier otro momento del pasado se vive una realidad positiva acerca del futuro de los métodos formales.

Esto se debe en parte a las necesidades por la cuales los investigadores consideraron los métodos formales: producir sistemas fiables de seguridad crítica [5, 6]. De hecho, actualmente y con el rápido crecimiento de los sistemas digitales esas necesidades son mayores que antes y los formalismos, de uno u otro tipo, continúan siendo la única solución real. Además, las cosas comienzan a cambiar: hace algunos años se consideraba a la Verificación formal de hardware como un prometedor tópico de investigación, pero no estaba listo para uso industrial; hoy, los métodos formales, particularmente el modelo de Verificación, se ganaron un espacio en la industria del hardware, por lo que su uso comienza a extenderse [7]. Gran parte de esta aceptación se debe a la rápida evolución en los tipos de métodos disponibles en la última década, muchos de los cuales hacen un trabajo mucho mejor para satisfacer las necesidades de la industria. Se espera que esta evolución continúe y que haya un proceso de selección natural. Por otra parte, las notaciones y los métodos que sobrevivan serán los que la industria acepte, simplemente porque aprovechan mejor los recursos que tienen a disposición.

Pero, ¿qué métodos y notaciones tendrán éxito? Para tratar de responder esta pregunta, en este artículo se describen algunos de los experimentos en métodos formales que diversos investigadores llevaron a cabo y lo que se ha aprendido de cada uno. Al final se hacen algunas observaciones acerca del perfil de lo que la industria necesita de la comunidad investigadora.

2. EXPERIENCIAS ANALIZADOS

A continuación se describen y analizan ocho experimentos que se encontraron en el proceso de investigación para este artículo.

2.1 Especificación formal en RAISE

Según Riesco *et al.* [8], gran parte de su interés en los métodos formales se remonta a la participación en el *MCC Formal Methods Transition Study* en 1990 y 1991 [9], ya que fue allí donde aprendieron del método RAISE y las herramientas desarrolladas en Europa, como parte del proyecto Espirit [10]. RAISE es sin duda uno de las técnicas de métodos formales más impresionante que existe. Integra conceptos de VDM, OBJ y CSP en una notación perfecta. Sin embargo, al momento de utilizarlo se encontraron muy pocas herramientas que apoyaran la verificación formal.

Como parte de sus primeros experimentos con métodos formales el quipo realizó la especificación de un micro *Real Time Executive* —µRTE— con la notación RAISE —RSL—, que implementaron principalmente en micro-código para mejorar su rendimiento. Si bien es muy simple, soporta la programación de tareas cíclicas y a-cíclicas y las sincronizaciones primitivas estándar, como bloqueo de exclusión mutua y descargas. Además, trataron de especificarlo en dos niveles de abstracción y de hacer pruebas informales de correctitud, para demostrar que a medida que la implementación es más detallada, la especificación es más abstracta.

A pesar de que lograron especificar el µRTE en RAISE, no todo fue éxito en el proyecto: a algunos de los ingenieros les gustaba el lenguaje RAISE, otros pensaron que era demasiado complejo, pero la mayoría simplemente no vio valor alguno en el ejercicio. Mientras unos estaban impresionados por lo sofisticación del lenguaje, también estaban preocupados por su complejidad y habría hecho falta demasiada formación para asegurar que todos los ingenieros lo usaran correctamente. Finalmente, las herramientas de análisis que RAISE proveía, al menos en ese momento, se limitaban a la sintaxis y a la comprobación de tipos. Las herramientas de verificación formal se habían planeado pero aún no estaban disponibles y, como resultado, los ingenieros no tenían mucha confianza en las pruebas informales de correctitud. Eventualmente, esta combinación de sintaxis compleja, la necesidad de una amplia formación y la falta de herramientas de análisis, los llevó a abandonar RAISE [11].

2.2 Verificación formal del AAMP5

Uno de los experimentos más grandes y mejor conocidos con métodos formales es la verificación formal de micro-código en el microprocesador AAMP5 [12-15]. Este proyecto fue patrocinado por Rockwell Collins y por NASA Langley y lo ejecutó SRI International y Rockwell Collins. El objetivo era evaluar la viabilidad de utilizar la verificación formal con PVS [16] para comprobar el micro-código del microprocesador.

El AAMP5 es un microprocesador fabricado por Rockwell que se utiliza ampliamente en productos Collins y como todos los procesadores AAMP tiene una arquitectura basada en pilas, un gran conjunto de instrucciones CICS y hace uso extensivo de micro-código [17]. Se diferencia de los anteriores modelos en que es segmentado y contiene unidades de procesamiento complejo, como la *Look Ahead Fetch* —LFU—. Contiene aproximadamente 500.000 transistores y proporciona un rendimiento equivalente entre un 386 y un 486 de Intel.

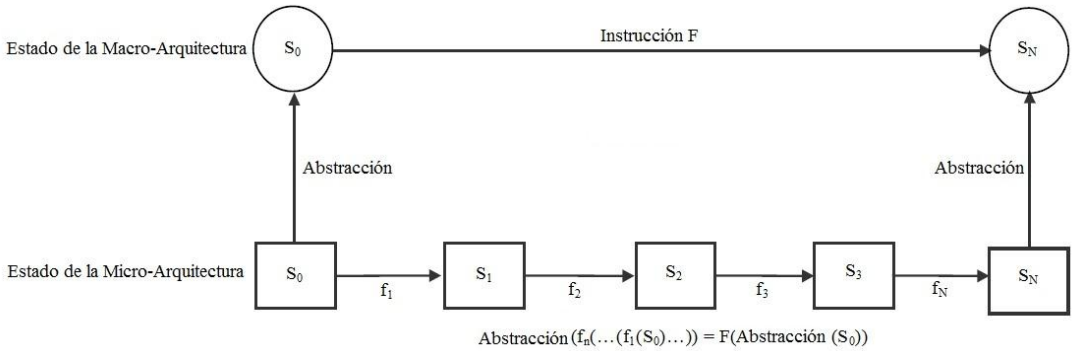

Fig. 1 Enfoque para la verificación del micro-código del AAMP5

El enfoque adoptado para este proyecto se representa en Figura 1. Se especificó formalmente al AAMP5 para los niveles de micro-arquitectura o registro de transferencia y la macro-arquitectura o conjunto de instrucciones; también se definió una función de abstracción para mapear los estados de la micro en la macro-arquitectura.

Probar el micro-código de una instrucción F correcta consiste en demostrar que el diagrama de la Figura 1 es conmutado, es decir, que a partir de micro-estado S_0 y aplicando las microinstrucciones f_1 a f_N seguidas por la función de abstracción y luego la macro-instrucción *F*, se obtiene algún macro-estado S_N.

La principal lección que se aprendió en este proyecto fue que era técnicamente posible demostrar la exactitud del micro-código. Más de la mitad de las instrucciones del AAMP5 se especificó en PVS, lo mismo que el diseño a nivel de transferencia de registros [18] y se demostró la exactitud de 11 instrucciones [19]. Además, se comprobó que los ingenieros pueden leer y escribir especificaciones formales. Mientras que la simplicidad del lenguaje PVS

desempeñó un importante rol en la comprobación, leerlo y escribirlo también estuvo a su alcance. Cualquier persona que pueda enseñar un lenguaje de programación moderno también puede enseñar un lenguaje de especificación formal, la cuestión está en la motivación no en la capacidad. Sin embargo, los ingenieros del proyecto nunca abandonaron su dominio específico, las notaciones gráficas, que es la forma como se especifican los procesadores actuales, lo que demuestra que estas presentaciones aportan el valor que falta en un lenguaje puramente textual [20].

Un problema menor encontrado en el proyecto fue la falta de variables de estado en PVS, porque la representación de estados fue algo que siempre tuvieron que evitar, lo que hacía que las especificaciones fueran más difíciles de leer. Mientras que la presencia de variables de estado puede hacer, y de hecho lo hacía, que las pruebas sean más difíciles, su ausencia además hace que las especificaciones sean menos intuitivas, levantando una barrera más para la aceptación. A medida que se creaba la especificación se encontraron dos errores reales en el micro-código del AAMP5, lo que convenció al equipo de que se alcanza un importante valor tan sólo por el hecho de escribir una especificación formal. Del mismo modo, se convencieron de que la verificación formal, por ejemplo las pruebas mecánicas de correctitud, proporciona un nivel muy alto de fiabilidad. Esto lo lograron al sembrar dos errores sutiles en el micro-código que entregaron al SRI y luego esperar para ver si los probadores los encontraban. SRI, efectivamente, los descubrió; pero lo que los impresionó fue el hecho de que este proceso fuera tan sistemático —no había forma de que no encontraron esos errores, excepto de que no hicieran las pruebas [21]—. Otra cosa que aprendieron fue la importancia de enfrentar problemas grandes y realistas; la razón de esto es que los ingenieros no siempre se enfrentan con problemas de esta escala y complejidad, hasta que se ven forzados a hacerlo. En este proyecto, algunos de los mayores problemas fueron cómo organizar las especificaciones, cómo abstraer los detalles irrelevantes, cómo estructurar pruebas complejas y cómo conseguir más velocidad de respuesta al PVS [22].

El mayor problema al que se enfrentaron fue su elevado costo, alrededor de 308 horas hombre por instrucción. Esta cifra es exagerada por una variedad de razones: la curva de aprendizaje era muy grande, el PVS evolucionaba a medida que se utilizaba, hubo necesidad de desarrollar muchas teorías de apoyo y, en lugar de tratar de verificar el mayor número de instrucciones, se intentó verificar instrucciones desde una variedad de clases. Era predecible que ese costo se podía reducir drásticamente la próxima vez, pero no se podía predecir en cuánto [23].

2.3 Verificación formal del AAMP-FV

Otro proyecto patrocinado por NASA y Rockwell Collins y dirigido por este último y SRI fue el AAMP-FV Microcode [24], cuyo objetivo era demostrar una reducción drástica de los costos mediante la reutilización de la infraestructura y la experiencia adquirida con el AAMP5.

El AAMP-FV fue un proyecto destinado específicamente a diseñar un procesador para utilizar con aplicaciones críticas, tales como aterrizaje de polo a tierra y alambrados para vuelo. Por esta razón fue más sencillo que el AAMP5: tiene cerca de 80 instrucciones y no es segmentado, aunque esto no simplificó se diseño; se calculó que si se fabricara industrialmente lo conformarían aproximadamente 100.000 transistores y tendría un rendimiento comparable a un 386 de Intel [25].

Algo que sorprendió al equipo de trabajo fue que la verificación formal de una instrucción AAMP-FV era comparable en dificultad a la de una instrucción AAMP5. Esto se debió a que no hicieron mucho uso de la abstracción en la especificación de su micro-arquitectura, como sí se hizo para el AAMP5. Aun así, lograron una reducción dramática en el costo para la mayoría de las instrucciones: completaron la verificación de 54 de las 80 establecidas con un costo cercano a 38 horas hombre por instrucción, aproximadamente una octava parte del costo del AAMP5. Además, estimaron que esto mismo se podría conseguir en 20 o quizás 10 horas por instrucción en un procesador similar. El punto clave fue que lograron la administración de las 54 aplicando un proceso repetible y bien definido. Ese tipo de previsibilidad es muy conveniente en la industria. Desafortunadamente, no todas las instrucciones del AAMP-FV se ajustaban a este proceso. Una veintena de ellas eran sustancialmente más complejas y las estrategias de prueba no funcionaron, lo que obligó a replantear de nuevo el proceso de investigación para esas instrucciones con costos considerablemente más altos.

Una de las principales lecciones aprendidas en este proyecto fue que cada dominio nuevo tiene una pronunciada curva de aprendizaje y que los costos se reducirán dramáticamente en el segundo o tercer intento. Desafortunadamente, esto también significa que no es posible sacar ninguna estimación significativa desde las pruebas iniciales, excepto la del comportamiento del peor de los casos [26]. Otra lección fue que la productividad requiere experiencia, tanto en el dominio del problema como en la metodología, porque la productividad del equipo despegó realmente cuando los expertos del dominio tenían experiencia suficiente en PVS, con el que estaban escribiendo las especificaciones para hacer las pruebas. La productividad requiere que alguien del equipo conozca ambos cuerpos de conocimiento.

Por primera vez la robustez de las pruebas se convirtió en una preocupación para todos. A menudo se encontraban revisando las especificaciones para completar hasta las pruebas más simples; lamentablemente, a menudo esto contradecía pruebas que se habían concluido meses antes. AAMP5 y AAMP-FV fueron proyectos inusuales en los que las pruebas eran muy estables pero, por el contrario, la norma en la industria es el cambio constante y es esencial que todos los análisis se automaticen para que sean lo suficientemente robustos como para acomodarse a estos cambios [13].

El proyecto AAMP-FV fue en muchos aspectos un gran éxito, sin embargo, se llevó a cabo por un pequeño número de ingenieros altamente capacitados, que poco a poco se dieron cuenta de que el uso generalizado de los métodos formales requeriría notaciones más intuitivas, incluso notaciones adaptadas a un dominio específico. Además, con tales notaciones también sabían que habría necesidad de hacer mayor énfasis en el método que en la notación.

2.4 El método CoRE

En los proyectos anteriores se demostró que el micro-código es una buena herramienta para explorar las pruebas formales de correctitud pero, debido a que los errores más importantes se presentan en la definición y especificación de los requisitos, este proyecto decidió centrarse en la modelización y el análisis de los mismos. Su objetivo fue demostrar que el modelo de requisitos puede ser más pequeño y manejable que el modelo de diseño.

Se comenzó por analizar el método CoRE [27, 28], que se basa en el modelo de cuatro variables de Parnas [29] y que representa y detalla una verdadera metodología, no sólo una notación. Una especificación CoRE consiste en un modelo único de requisitos con dos aspectos: el modelo de comportamiento que especifica la funcionalidad del sistema y el modelo de clases que es un paquete del modelo de comportamiento que proporciona una forma para estructurar las especificaciones grandes y cómo planificar y gestionar los cambios. Para validar el método se trabajó sobre la lógica del modo de control de un sistema de dirección de vuelos [3, 30]. Esta lógica es una de las partes más complejas de estos sistemas y, para mantener el proyecto dentro de lo que se buscaba, el equipo tuvo que hacer simplificaciones, como eliminar algunos de los modos más complicados de operación, no especificar las interfaces de hardware y no ocuparse de las fallas de los componentes internos. Aun así, fue un proyecto grande que clasifica como estudio de casos industriales reales y que tomó 560 horas hombre, cerca de nueve meses, para su culminación; posteriormente varios investigadores se dedicaron a trabajar con él [31-33].

El equipo de trabajo no contaba con herramientas para crear la especificación CoRE, por lo que la validaron a través de una serie de inspecciones con expertos en control de vuelos y algunos de sus ingenieros de software. Estas inspecciones requirieron un total de 35 horas hombre y se encontraron cerca de 100 errores en la especificación que requirieron más de 300 horas para corregirlos. Esto indica la efectividad de las inspecciones: son económicas, encuentran muchos defectos y son una excelente manera de formar a los nuevos miembros del equipo. Incluso, cuando se hicieron las pruebas de correctitud de micro-código, se encontró que las inspecciones eran una las técnicas más útiles, así que una lección obvia fue que las notaciones y herramientas formales deben soportar las inspecciones. Sin embargo, frecuentemente las herramientas de modelado ofrecen formas adecuadas para imprimir el modelo y, en casos muy extremos, se tiene que llevar a cabo inspecciones de las capturas de pantalla. Por todo esto es obvio y se entiende la importancia de las inspecciones en la práctica industrial. La principal experiencia que dejó este proyecto fue que es posible especificar los requisitos mediante modelos formales o matemáticos. Dado que el estado actual de la práctica es especificar los requisitos en inglés, con unas pocas tablas y diagramas, este es un gran cambio cuya aceptación no se espera a corto plazo.

El modelo de clases de CoRE enfatiza en cuestiones de arquitectura y abarca desde los requisitos, el diseño y el código, hasta la prueba. Gran parte de la arquitectura del sistema la define el problema y uno de los mayores desafíos es calcular lo que verdaderamente es. Posiblemente, durante las inspecciones se lean bien las especificaciones CoRE, pero puede que no ayuden a ver los problemas reales, como cuando se define una arquitectura equivocada. Para corregir esos errores se debe aplicar un esfuerzo considerable, pero con seguridad que el equipo se ahorrará gran cantidad de problemas si identifica la arquitectura correcta antes de empezar y, finalmente, llegará a ver el modelo de clases tan completo como el de comportamiento. De hecho, podrá encontrar que necesitan más construcciones arquitectónicas y aspectos como replicación, parametrización e incluso herencia.

El equipo encontró, además, que el modelo de cuatro variables puede ser un paradigma útil y una y otra vez tuvieron que escribir y discutir algunos puntos clave de la especificación. Aprendieron, una vez más, que los ingenieros prefieren notaciones gráficas, debido a que son más fáciles de leer que el texto y a que ayudan a hacer los formalismos más aceptables [34]. Mientras CoRE se basa en un modelo formal, carece de sintaxis y semántica formales. Recapitulando en el proyecto, sorprende lo lejos que llegaron sin una semántica de este tipo, lo que ilustra la importancia en el diseño de software de muchos aspectos "*informales*", tales como las cuestiones arquitectónicas discutidas anteriormente. Sin embargo, finalmente llegaron a un punto en el que todos estuvieron de acuerdo en que necesitaban tanto una sintaxis como una semántica formales.

Por último, la falta de herramientas se convirtió a la vez en una debilidad y una fortaleza, porque mientras que crear la sintaxis y comprobar los tipos manualmente era muy tedioso y propenso a errores, no usar una herramienta les permitió la libertad para improvisar cuando fuera necesario. Utilizar una herramienta también implica comprometerse con todas las restricciones que impone, por lo que es muy importante tener una clara comprensión de lo que se trata de lograr y asegurarse de que la herramienta es compatible con esos objetivos. No usar una herramienta cualquiera es generalmente mejor que usar una herramienta equivocada.

2.5 La especificación formal SCR

Este proyecto utilizó la herramienta SCR del Naval Research Lab [35, 36] que aplicaron en el avión A-7E. Si bien SCR carece de la estructura de clases encontrada en

CoRE, sí tiene sintaxis y semántica formales que le permiten soportar una variedad de análisis automatizados. Además, sus modelos se pueden ejecutar utilizando el simulador proporcionado en la herramienta.

Dada la similitud entre SCR y CoRE, trabajar en la especificación del sistema de control de vuelos del A-7E con esta herramienta fue más sencillo y al equipo le llevó un poco más de 100 horas hombre hacerlo. Debido al esfuerzo que se invirtió en la inspección y revisión de la especificación en CoRE, no se esperaba encontrar muchos errores en ésta. En total encontraron 27 errores en el modelo original de SCR, algunos de ellos importantes. El desglose de estos errores se muestra en la Tabla 1.

Tabla 1 Errores encontrados usando SCR

Encontrado por	**Severidad del error**				
	Trivial	Menor	Moderado	Mayor	Total
Modelo de creación SCR		7	4	1	12
Herramienta de análisis SCR			2	2	4
Modelo de ejecución SCR		1	3	3	7
Otras herramientas	2	1	1		4
Total	2	9	10	6	27

Los errores triviales fueron de ortografía y puntuación; los menores aquellos que definitivamente serían capturados durante la implementación; los moderados los que probablemente serían capturados durante el desarrollo, pero que podrían perderse y los mayores los sutiles que probablemente no serían capturados durante la implementación y tendrían un efecto significativo en el producto final.

Aunque no son muchas las conclusiones que se pueden extraer de una muestra de sólo 27 errores, existen algunas tendencias: simplemente con introducir el ejemplo en la herramienta se encontraron más errores, aunque leves o moderados y los errores detectados por las herramientas de análisis SCR fueron más significativos, lo que se esperaba. Los siete errores encontrados con el modelo de ejecución se convirtieron en una sorpresa, porque fue el proceso se realizó de forma *ad hoc*; uno de ellos fue un error menor, dos fueron encontrados por las herramientas de análisis, excepto que el análisis tomó demasiado tiempo y se desactivó antes de terminar. Tres de ellos fueron malentendidos por parte del equipo de trabajo al no interpretar adecuadamente la semántica de SCR y uno fue un error significativo, también del equipo.

La lección obvia aquí es que los análisis automatizados encuentran errores que los métodos manuales no logran identificar. Diversos investigadores han encontrado esto mismo en otros proyectos [36, 37] y probablemente puedan aceptarlo sin controversia. Por otra parte, todavía no es posible renunciar a las inspecciones manuales, porque son necesarias para aplicar en casos en los que a las herramientas no les va tan bien, como la comprobación del estilo y la facilidad de mantenimiento.

Para los ingenieros en este proyecto fue una sorpresa que la simulación encontrara varios de los errores, particularmente los sutiles malentendidos semánticos. Mientras que esto refuerza el valor de la simulación como una técnica de validación, también es un poco desalentador, debido a que la simulación tiene el mismo problema de las pruebas: no tiene un criterio bien definido para parar, es decir, no es fácil saber cuándo se ha hecho lo suficiente. Pero la mayor sorpresa se presentó cuando conectaron la simulación a una maqueta de la cabina de vuelo, en la que podían presionar botones, observar las luces indicadoras y al mismo tiempo ejecutar el modelo de requisitos. Entonces, el modelo de requisitos se convirtió en un prototipo rápido, por lo que el equipo aceptó su efectividad casi de inmediato y se convirtió en una importante lección: conectar un modelo ejecutable a una maqueta de la interfaz de usuario hace accesible el formalismo para ingenieros y clientes.

2.6 Especificación formal con T-VEC

La especificación lógica del anterior simulador de vuelos también se utilizó para evaluar el generador de vectores de prueba T-VEC, un entorno que genera casos de prueba y resultados esperados a partir de una especificación formal del comportamiento funcional de un sistema. También comprueba las inconsistencias y las guarda como especificaciones que no se pueden probar [38, 39].

Para este experimento se utilizó un traductor T-VEC, desarrollado por el Software Productivity Consortium, para generar una especificación de prueba a partir del modelo lógico de la especificación, combinado con las especificaciones escritas manualmente del mismo traductor para describir desde las variables de entrada y de salida del software de bajo nivel, hasta las más abstractas del SCR. La especificación completa del T-VEC se utilizó para generar los casos de prueba y su cobertura, evaluados en una implementación Java del modelo lógico. Si bien muchos aspectos de los resultados todavía tienen que ser investigados, estas primeras pruebas alcanzaron un alto nivel de condiciones múltiples de cobertura del código Java. La herramienta T-VEC también identificó varios errores en el modelo SCR y en este código.

2.7 Detección de modos de confusión

Otro experimento patrocinado por NASA en métodos formales tenía como objetivo determinar si los métodos formales se pueden utilizar para detectar fuentes de modo de confusión en un sistema automatizado. El modo de confusión es una situación en la que un operador humano puede confundirse acerca del estado correcto de la automatización o tiene un modelo mental incorrecto de cómo se comporta [40, 41], algo que ha jugado un rol importante en algunos accidentes aéreos [42, 43].

Debido a que es posible especificar formalmente el comportamiento de la automatización ¿qué fuentes de modos de confusión se pueden caracterizar formalmente para buscarlas con herramientas automatizadas? Por ejemplo, ¿existe algún estado del sistema en el que se pase por alto una orden directa de la tripulación del vuelo? En este proyecto se especificó en PVS el modo lógico del Flight Guidance System, sin embargo, en esa ocasión se cambió la arquitectura para apoyar de mejor forma a la familia completa del sistema. Este proceso originó muchas piezas pequeñas y componentes cohesionados, muy similar a la arquitectura que se obtendría al aplicar un enfoque orientado por objetos. Esta arquitectura es muy superior a la que se desarrolló en la especificación original de CoRE.

Debido a que se utilizó un sistema de especificación formal, fue muy natural para el equipo declararlos como lemas que se pudieron demostrar con PVS. Las pruebas también fueron muy simples y normalmente requerían sólo unos pocos comandos en el lenguaje. En poco tiempo se desarrolló una suite completa de pruebas que, cada vez que se cambiaba algo significativo en el modelo originaba una nueva ejecución. Muy a menudo, una prueba fallaba indicando que algo estaba equivocado, lo que resultó muy útil para la construcción de los modelos porque las áreas de producción rutinariamente realizaban pruebas de regresión después de hacer un cambio. De manera similar, a causa de que se automatizaron algunas de las verificaciones que normalmente se harían manualmente, fue posible, de forma sencilla y a bajo costo, repetir las suites de regresión para estos análisis.

La mayor sorpresa en este proyecto fue que el equipo se encontró ejecutando pruebas, no porque estuvieran tratando de evaluar una utilidad o de demostrar un punto determinado, sino porque estaban ayudando a hacer el trabajo. Uno de los debates que continúa en la comunidad de los métodos formales es si los ingenieros deben hacer pruebas pero, después de este proyecto, parece quedar claro que esta actividad seguirá integrada naturalmente a la creación de modelos, ofreciendo pruebas del sistema integradas con el entorno de modelado lo suficientemente potentes como para que no sean demasiado costosas.

Finalmente, se logró algo de éxito en la automatización de la detección de ciertas fuentes de modos de confusión en los modelos, que se detalló en el reporte final de la fase I [44]. También se descubrió la necesidad de investigar si la capacidad de ocultar información de PVS se puede utilizar para hacerla cumplir de manera explícita.

2.8 Especificación formal del JEM

El último experimento que se discute en este trabajo es un excelente ejemplo de cómo insertar los métodos formales en un proceso industrial existente. El microprocesador JEM es la primera ejecución de *Java Virtual Machine* directamente en hardware [45]. Dos ingenieros del proyecto crearon una especificación formal en ACL2 de la micro-arquitectura del JEM [46] que ejecuta micro-código JEM [47]. Pero lo más importante fue que optimizaron la especificación ACL2 para que se ejecutara más rápido en el simulador que el código C original. De hecho, reemplazaron ese código con la versión ACL2 y demostraron que es posible conseguir los mismos resultados que cuando se ejecuta la suite de pruebas del JEM. La importancia de esto fue que encontraron una forma para sustituir un modelo tradicional con un modelo formal construido sin costo adicional. Ahora que se tiene el nuevo modelo, las pruebas se ejecutan naturalmente y se han probado varias propiedades importantes de la micro-arquitectura JEM.

3. LAS NECESIDADES DE LA INDUSTRIA

A continuación se describe un perfil de lo que la industria necesita de los métodos formales, que se ha determinado a partir de los resultados obtenidos en los proyectos.

- Especificaciones ejecutables. En la comunidad de los métodos formales se vive un considerable debate acerca de si las especificaciones deben ser ejecutables. La preocupación es que son demasiado operacionales e insuficientemente abstractas. Pero las experiencias con la herramienta SCR convencieron a los diferentes equipos de que es posible escribir especificaciones ejecutables útiles y los logros que hasta el momento se han alcanzado son una buena prueba para proseguir. A través de la integración de un modelo ejecutable con una simulación de la interfaz de usuario, se obtiene un prototipo rápido y temprano en el ciclo de vida, que se puede utilizar para revisar los requisitos del cliente [48]. Una advertencia aquí es que la velocidad puede ser importante, por ejemplo, si la especificación ACL2 de JEM corriera sólo a la mitad de lo que lo hace el código en C, no habría sido una opción viable.

- Claridad y legibilidad de las especificaciones, porque luego de analizar los resultados todavía continúan como el talón de Aquiles de los métodos formales. Para hacerle frente a cada proyecto se necesita mejores formas para documentar el bosque antes que los árboles y se requiere integrar notaciones gráficas y textuales lo mismo que formas intuitivas, por lo que las notaciones necesitan ser tan simples como sea posible. En algunos casos es posible tener notaciones específicas para cada dominio particular.

- Soporte para la extensión, los cambios y la reutilización. Se requieren mecanismos para organizar y hacer legibles especificaciones extensas. Los cambios es la única constante que es posible medir en la industria, por lo que es necesario planificarlos y gestionarlos. Dado que usualmente se construyen variaciones de los mismos sistemas una y otra vez, la industria debe lograr la reutilización masiva de especificaciones para cada nuevo producto.

- Con frecuencia se ofrece notaciones y herramientas sin los métodos necesarios, por lo que es necesario comprender los conceptos fundamentales que realizan el trabajo de aproximación y los pasos para aplicarlos. También ayudaría contar con estudios de casos industriales que sirvan como ejemplo.

- Uno de los mayores obstáculos para insertar cambios en la industria es la incompatibilidad con las prácticas existentes. Existen cientos de prácticas que se aplican en la producción de un producto y, sólo es posible cambiar más que unas pocas en el proyecto. Por ejemplo, es muy difícil incorporar una herramienta grande y monolítica a las prácticas actuales, pero si se pudiera dividir en sus componentes primarios, probablemente los ingenieros encontrarían formas, que ni siquiera se han pensado, para utilizar esas piezas. Estos mismos modelos también se pueden utilizar para el desarrollo, la ejecución y el análisis, porque pocas veces existe tiempo para crear un modelo para especificación, otro para simulación y otro para verificación. Los paradigmas orientados por objetos han ganado popularidad suficiente en la última década, lo que podría proporcionar una vía para implementar los métodos formales en la industria [49]. En particular, es necesario que se realicen más proyectos e investigaciones acerca de cómo implementar la semántica formal en UML.

- Con la posible excepción de la especificación ejecutable, ninguna de las anteriores necesidades requiere una semántica formal, lo que refleja que muy pocas de las actuales ameritan métodos formales. Por esta misma razón es que se requieren herramientas con modelos ejecutables, aunque sin una semántica bien definida, para que se conviertan en aportes a la industria y abonen el camino para la aceptación de estos métodos en tiempos cercanos. Mientras se espera que esto ocurra, la industria debe entender que, en algunos aspectos, son un callejón sin salida [50]. La razón es que el siguiente paso, más allá de las especificaciones ejecutables, es el análisis con herramientas automatizadas. Se debe ser capaz de lograr la consistencia simple y la comprobación de completitud para demostrar las propiedades específicas de aplicación, particularmente las de seguridad. Pero todo esto requiere disciplina y aplicación constante de una notación, porque cuanto más simple y más cuidadosamente se piensen los análisis más lejos son capaces de conducir.

- Es importante no perder de vista a las pruebas. En la actualidad se invierte entre una y dos terceras partes del presupuesto del proyecto a ellas. La generación automática de casos de prueba desde un modelo formal podría casi justificar el costo de la construcción del modelo; inclusive, sería mejor si fuera posible encontrar cómo reemplazar algunas pruebas por análisis.

- Otro de los debates en la comunidad de los métodos formales es que los ingenieros puedan realizar pruebas rutinariamente, pero las experiencias descritas con los modelos PVS y el modelo lógico del simulador de vuelos pueden hacerlos cambiar de opinión, porque ellas surgen como una consecuencia natural de la construcción de modelos cuando están integradas en el entorno del modelado. La clave es conseguir que los ingenieros se atrevan a construir modelos y que se convenzan de que requieren notaciones y entornos naturales e intuitivos para ellos, no para los expertos en métodos formales.

4. CONCLUSIONES

Uno de los descubrimientos verdaderamente interesantes de los últimos años fue que los dinosaurios nunca se extinguieron realmente, que rodean todo en la naturaleza y que se conocen como aves. De manera similar, se puede sugerir que los métodos formales no están en peligro de extinción, porque la industria los utiliza de forma generalizada y rodean todo en las Ciencias Computacionales, donde se conocen como álgebra de Boole, máquinas de estado, gramáticas,... Entonces, la cuestión no es si los métodos formales serán utilizados por la industria en el futuro, debido a que actualmente lo está haciendo y con seguridad así continuarán, la verdadera pregunta es cuál de las "especies" más grandes de los métodos formales se desarrollará lo suficiente como para sobrevivir.

5. REFERENCIAS

[1] Craigen, D.; Gerhart, S. & Ralston, T. (1993). An International Survey of Industrial Applications of Formal Methods, Vol. 2 Case Studies. NRL/FR/5546--93-9582. Naval Research Laboratory, USA.

[2] Bowen, J. & Hinchey, M. (1995). Applications of Formal Methods. Prentice-Hall International Ltd.

[3] Miller, S. (1998). Specifying the Mode Logic of a Flight Guidance System in CoRE and SCR. Proceedings Second Workshop on Formal Methods in Software Practice, pp. 44-53. Clearwater Beach, Florida, USA.

[4] Ciapessoni, E. et al. (1999). From Formal Models to Formally Based Methods: An Industrial Experience". ACM Transactions on Software Engineering and Methodology, 8(1), pp. 79-113.

[5] Butler, R. & Finelli, G. (1993). The Infeasibility of Experimental Quantification of Life-Critical Software Reliability. IEEE Transactions on Software Engineering, 16(5), pp. 66-76.

[6] Littlewood, B. & Strigini, L. (1993). Validation of Ultra-High Dependability of Software-Based Systems. Communications of the ACM, 36(11), pp. 69-80.

[7] Serna, M.E. (2010). Formal Methods and Software Engineering. Revista Virtual Universidad Católica del Norte, 30, pp. 158-184.

[8] Riesco, D. et al. (2005). Using a feature model for RAISE specification reusability. Proceedings IEEE International Conference on Information Reuse and Integration, pp. 306-311. Las Vegas, USA.

[9] Gerhart, S.; Ralston, T. & Russinoff, D. (1991). Formal Methods Transition Study. Final Report, Technical Report STP-FT-322-91. Microelectronics and Computer Technology Corporation, USA.

[10] George, C. (1991). The RAISE Specification Language: A Tutorial. Lecture Notes in Computer Science, 552, pp. 238-319.

[11] Austin, S. & Parkin, G. (1993). Formal methods: A survey. Technical report, National Physical Laboratory. Teddington, Middlesex, UK.

[12] Miller, S. & Srivas, M. (1995). Formal Verification of the AAMP5 Microprocessor. Proceedings 1th Workshop on Industrial-Strength Formal Specification Techniques, pp. 2-16. Boca Raton, Florida, USA.

[13] Srivas, M. & Miller, S. (1995). Formal Verification of the AAMP5 Microprocessor : a case study in the industrial use of formal methods. In Bowen, J. & Hinchey, M. (Eds.), Applications of Formal Methods. Prentice-Hall International Ltd.

[14] Srivas, M. & Miller, S. (1995). Applying Formal Verification to a Commercial Microprocessor. Proceedings IFIP Conference on Hardware Description Languages and Their Applications, pp. 493-502. Makuhari, Japan.

[15] Srivas, M. & Miller, S. (1995). Formal Verification of an Avionics Microprocessor. Technical Report, NASA-95-cr4682. Hampton, Virginia, USA.

[16] Owre, S. et al. (1995). Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS. IEEE Transactions on Software Engineering, 21(2), pp. 107-125.

[17] Best, D. et al. (1982). An Advanced-Architecture CMOS/SOS Microprocessor. IEEE Micro, 2(4), pp. 11-26.

[18] Clarke, E.M. & Wing, J.M. (1996). Formal methods: State of the art and future directions. ACM Computing Surveys, 28(4), pp. 626-643.

[19] Bowen, J. et al. (1996). An invitation to formal methods. IEEE Computer, 29(4), pp. 18-19.

[20] Glass, R.L. (1996). Formal methods are a surrogate for a more serious software concern. IEEE Computer, 29(4), pp. 16-17.

[21] Woodcock, J. et al. (2009). Formal Methods: Practice and Experience. ACM Computing Surveys, 41(4), pp. 1-40.

[22] Miller, S. (1998). The industrial use of formal methods: was Darwin right? Proceedings Second IEEE Workshop on Industrial Strength Formal Specification Techniques, pp. 74-83. Boca Raton, Florida. USA.

[23] Bloomfield, R. & Craigen, D. (1999). Formal methods diffusion: Past lessons and future prospects. Technical Report D/167/6101, Adelard, Coborn House. London E3 2DA, UK.

[24] Miller, S. et al. (2003). Formal Verification of the AAMP-FV Microcode. Final Report, Technical Report: NASA-99-cr208992.

[25] Börger, E. & Stärk, R. (2003). Abstract State Machines: A Method for High-Level System Design and Analysis. Springer-Verlag.

[26] Bush, W.R.; Pincus, J.D. & Sielaff, D.J. (2000). A static analyzer for finding dynamic programming errors. Software - Practice and Experience, 30(7), pp. 775-802.

[27] Faulk, S. et al. (1992). The CoRE Method for Real-Time Requirements. IEEE Software, 9(5), pp. 22-33.

[28] Faulk, S. et al. (1994). Experience Applying the CoRE Method to the Lockheed C-130J Software Requirements. Proceedings Ninth Annual Conference on Computer Assurance, pp. 3-8. Gaithersburg, USA.

[29] Miller, S. & Tribble, A. (2001). Extending the four-variable model to bridge the system-software gap. Proceedings of the 20th Digital Avionics Systems Conferene (DASC01), pp. 1-12. Daytona Beach, Florida, USA.

[30] Miller, S. & Hoech, K. (1997). Specifying the Mode Logic of a Flight Guidance System in CoRE. Technical Report WP97-2011. Rockwell Collins, Information Center.

[31] Jeffords, R. & Heitmeyer, C. (2001). An Algorithm for Strengthening State Invariants Generated from Requirements Specifications. Proceedings of the Fifth IEEE International Symposium on Requirements Engineering, pp. 182-192. Toronto, Canada.

[32] Miller, S. (2003). Proving the Shalls: Requirements, Proofs, and Model-Based Development. Proceedings 12th International FME Symposium Formal Methods Europe (FME2003), pp. 261-280. Pisa, Italy.

[33] Constance, H. (2004). Managing Complexity in Software Development with Formally Based Tools. Electronic Notes in Theoretical Computer Science, 108(13), pp. 11-19.

[34] Ghose, A. (2000). Formal methods for requirements engineering. 2000 International Symposium on Multimedia Software Engineering (ISMSE 2000), pp. 13-16. Taipei, Taiwan.

[35] Heitmeyer, C. et al. (1995). SCR*: A Toolset for Specifying and Analyzing Requirements. Proceedings Tenth International Conference on Computer Assurance, pp. 109-122. Gaithersburg, USA.

[36] Heitmeyer, C.; Jeffords, R.D. & Labaw, B. (1996). Automated Consistency Checking of Requirements Specification. ACM Transactions on Software Engineering and Methodology (TOSEM), 5(3), pp. 231-261.

[37] Heimdahl, M. & Leveson, N. (1996). Completeness and Consistency in Hierarchical State-Based Requirements. IEEE Transactions on Software Engineering, 22(6), pp. 363-377.

[38] Blackburn, M. & Busser, R. (1996). T-VEC: a Tool for Developing Critical Systems. Proceedings Eleventh Annual Conference on Computer Assurance, pp. 237-249. Gaithersburg, USA.

[39] Blackburn, M. (1997). Specification Transformation and Semantic Expansion to Support Automated Testing. Ph.D. Dissertation. Information Technology, George Mason University.

[40] Leveson, N. (1995). Safeware: System Safety and Computers. Addison-Wesley Publishing Company.

[41] Leveson, N. et al. (1997). Analyzing Software Specifications for Mode Confusion Potential. Proceedings Workshop on Human Error and System Development, pp. 1-16. Glascow, UK. 1997.

[42] Hughes, D. & Dornheim, M. (1995). Automated Cockpits: Who's in Charge? Parts 1 & 2. Aviation Week & Space Technology, 5-6, pp. 142.

[43] Billings, C. (1997). Aviation Automation: The Search for a Human-Centered Approach. Lawrence Erlbaum Associates.

[44] Miller, S. & Potts, J. (1999). Detecting Mode Confusion Through Formal Modeling and Analysis: Phase I Final Report. Technical Report: NASA-99-cr208971. Langley Research Center.

[45] Greve, D. & Wilding, M. (1998). Stack-Based Java a Back-to-Future Step. Electronic Engineering Times, Jan. 12, pp. 92-94.

[46] Boyer, R.S. & Moore, J.S. (1998). A Computational Logic. New York: Academic Press.

[47] Wilding, M.; Greve, D. & Hardin, D. (1998). Efficient Simulation of Formal Processor Models. Formal Methods in System Design, 18(3), pp. 233-248.

[48] Kelley, S.A. & Clarkson, M. (2002). Formal Methods Application: An Empirical Tale of Software Development. IEEE Transactions on Software Engineering, 28(3), pp. 308-320.

[49] Bowen, J. & Hinchey, M. (1997). The Use of Industrial-Strength Formal Methods. Proceedings 21st International Computer Software and Applications Conference, pp. 332-337. Washington, USA.

[50] Clarke, E.M. et al. (2005). Model checking: Back and forth between hardware and software. Proceedings First IFIP TC 2/WG 2.3 Conference on Verified Software: Theories, Tools, Experiments, pp. 251-255. Zurich, Switzerland.