Métodos Formales y Tecnologías Orientadas a Objetos

José Nelson Pérez Castillo

RESUMEN

Este artículo discute el uso de métodos formales en procura de los más altos niveles de calidad en el desarrollo de software. Se muestra un panorama de los principales conceptos de las tecnologías orientadas a objetos y de los métodos formales haciendo un recorrido de la investigación y las tendencias principales en este importante campo de la ingeniería de software.

Palabras claves: Métodos formales, Z++, VDM++, Object-Z, tipos, categorías, tecnologías orientadas a objetos.

FORMAL METHODS AND OBJECT TECHNOLOGY

This paper reviews the use of formal approaches in securing better quality software. It overviews the main concepts of the object techonology and formal methods, and conduct a brief survey of research into the principal strands in this important field of the software engineering.

Keywords: Formal methods, Z++, VDM++, Object-Z, types, category theory, object technology

INTRODUCCIÓN

El estudio sistemático y uso intensivo de métodos formales en la ingeniería de software, es una disciplina que se ha abordado hace ya varios años en la Facultad de Ingeniería de la Universidad Distrital, específicamente en la Maestría en Teleinformática, en donde se les conoce como Técnicas de Especificación Formal para software de comunicaciones, siendo las más representativas: ASN.1, SDL y LOTOS.

El incentivo fundamental para el trabajo de investigación sobre métodos formales en la ingeniería de software, fue la creciente certeza de que únicamente los enfoques formales en la especificación, verificación, análisis, implementación, prueba y operación de los sistemas informáticos, podría proporcionar los medios para controlar la complejidad creciente de los sistemas de información actuales y de la interconexión de sistemas abiertos. Esto significa que los métodos formales, se están convirtiendo en una herramienta de uso común para todos aquellos involucrados en el ciclo de desarrollo de sistemas de alta calidad, lo que implica que tales métodos formales tengan que ser estables, cuestión que puede garantizarse únicamente mediante su estandarización. El CCITT y la ISO proporcionan el ambiente natural para dicho proceso [1].

RAZONES EN PRO DE LOS MÉTODOS FORMALES

Los métodos formales, son el resultado del trabajo de varias décadas sobre lenguajes de especificación formal y métodos rigurosos para el

Los métodos formales, son el resultado del trabajo de varias décadas sobre lenguajes de especificación formal y métodos rigurosos para el desarrollo de sistemas de computación.

Los orígenes de la investigación sobre métodos formales, se remontan a la década de los setenta, con los trabajos de Dijkstra y Hoare sobre verificación y desarrollo de programas y los de Scott, Stratchey y otros sobre semánticas de programas.

desarrollo de sistemas de computación. La esencia de tales lenguajes y métodos la constituye un soporte matemático que busca asegurar precisión y tratabilidad. Las descripciones convencionales, generalmente se dan en lenguaje natural o en diagramas, pero es complicado hacer que estas no sean ambiguas y por tanto difíciles de analizar. Los errores y las omisiones en los sistemas de computación son costosos de rectificar y pueden poner en peligro su vida útil y apropiado funcionamiento [2]. Las técnicas formales se justifican particularmente, para sistemas con los siguientes atributos:

Complejos: muchos sistemas, de por sí son complejos y la tendencia es producir sistemas cada vez con mayor complejidad.

Concurrentes: son sistemas que exhiben patrones complejos de comportamiento y potencialmente interferentes que deben ser entrelazados. La concurrencia surge en sistemas distribuidos, sistemas en tiempo real, diseño de hardware y procesamiento en paralelo.

Calidad crítica: son sistemas cuyas fallas no son peligrosas, pero cuya confiabilidad es altamente importante; ejemplos de esto son las aplicaciones financieras, las de telecomunicaciones y los sistemas operativos.

Protección crítica: son sistemas que controlan actividades vitales tales como la defensa, la medicina, la industria nuclear, las telecomunicaciones y la gestión del tráfico aéreo.

Seguridad crítica: con el uso extendido de las tecnologías computacionales puede ser esencial impedir el uso no autorizado de información o facilidades de computación, por motivos de seguridad y confidencia comercial o privacidad personal.

Estandarizados: los estándares, en particular los internacionales, suelen ser ampliamente utilizados y deben ser interpretados uniformemente, si se quiere que tengan algún valor.

CONCEPTUALIZACIÓN DE LOS DENOMINADOS MÉTODOS FORMALES

Los orígenes de la investigación sobre métodos formales, se remontan a la década de los setenta, con los trabajos de Dijkstra y Hoare sobre verificación y desarrollo de programas y los de Scott, Stratchey y otros sobre semánticas de programas. Un avance importante lo constituyó el desarrollo de lenguajes matemáticos para la verificación de programas, como lo fueron las notaciones Z [3] y VDM [4]. El lenguaje Z fue creado por J. R. Abrial, luego fue expandido y aplicado industrialmente por otros desarrolladores durante la década de los ochenta, gracias al Grupo de Investigación en Programación de Oxford, que hizo de los métodos formales un elemento clave de su trabajo. Por su parte, el lenguaje VDM fue desarrollado inicialmente en los laboratorios de IBM en Viena y por el grupo de la Universidad de Manchester liderado por Cliff Jones, quien los convirtió en el centro de atención de sus investigaciones y esfuerzos por lograr instrumentos de desarrollo, que condujeron al logro de la herramienta conocida como Mural [5] y al proceso de estandarización adelantado por la ISO [6].

Tanto Z como VDM, son lenguajes de especificación formal "basados en modelos", es decir, utilizan la teoría de conjuntos y la lógica, para construir modelos abstractos de los sistemas que se requieran, mediante el uso de conjuntos, secuencias, funciones, etc.

En contraste, los métodos "algebraicos" tales como OBJ, PLUSS [7], LARCH [8] y FOOPS [9] usan lógica de ecuaciones para proveer descripciones de sistemas implícitas y abstractas.

Las álgebras de procesos, constituyen el fundamento de los métodos formales orientados a la solución de problemas de concurrencia. Los principales formalismos son CSP [10], CCS [11] y el cálculo-π [12]. El fundamento de estas álgebras es el concepto abstracto de "proceso",

definido como un componente que puede reaccionar a eventos externos, generarlos y realizar el procesamiento interno requerido. Cada lenguaje proporciona los significados a procesos específicos, los combina y razona respecto a sus propiedades. Aplicaciones importantes de CSP incluyen el desarrollo de la unidad de punto flotante INMOS T800 [13] y el lenguaje OCCAM.

Los lenguajes CCS y el cálculo-π, han servido como vehículos para un trabajo teórico extenso. Las ideas de las álgebras de procesos "puras", también han contribuido al desarrollo de lenguajes híbridos como LOTOS [14], OBJSA [15] y RSL [16], en donde un componente de álgebra de procesos se combina con un lenguaje de especificación de datos (algebraico en el caso de LOTOS, tanto algebraico como basado en modelos en el caso de RSL). LOTOS se usa ampliamente en el campo de las telecomunicaciones y RSL por su parte ha sido el tema de proyectos de industrialización a gran escala, financiados por la Unión Europea.

Los formalismos basados en modelos, se usan ampliamente de manera conjunta con las tecnologías orientadas a objetos, a través de lenguajes tales como Object-Z [17], VDM++ [18] y métodos tales como Syntropy [19] (que usa notación Z) y Fusion [20] (relacionado a VDM). Si bien estos formalismos son efectivos en la modelación de estructuras de datos, como conjuntos y relaciones entre conjuntos, no son los más idóneos para la captura de mecanismos sofisticados como son el enlace dinámico y el polimorfismo, en tecnologías orientadas a objetos.

Cada lenguaje proporciona los significados a procesos específicos, los combina y razona respecto a sus propiedades.

Los formalismos algebraicos se han usado menos frecuentemente con las tecnologías de objetos. El ejemplo más importante de tal combinación es el lenguaje FOOPS y el formalismo OOZE [21] en el que la lógica de ecuaciones se combina con un mecanismo de estructuración de módulos. Los aspectos dinámicos de los sistemas de objetos, tales como el conjunto de objetos existentes de una clase, se manejan directamente a través del concepto de "metaclase" para cada clase y los métodos de la metaclase. El polimorfismo se trata a través de la jerarquía de tipos.

El álgebra de procesos, también puede usarse de modo orientado a objetos de la manera como se indica en [22]. Más recientemente, el cálculo-π se utilizó para dar una base semántica a una extensión orientada a objetos de VDM [23]. Las álgebras de procesos son muy efectivas en la captura de algunos aspectos dinámicos de sistemas de objetos, pero el tratamiento de datos complejos requiere, bien sea un lenguaje de especificación adicional para estos datos o una codificación altamente compleja de datos a manera de procesos, como en el cálculo-π

MÉTODOS FORMALES EN TECNOLOGÍAS DE OBJETOS

Los seguidores de los métodos formales y los expertos en tecnologías de objetos, investigan actualmente de qué manera las especificaciones formales pueden complementar el desarrollo orientado a objetos o por lo menos, cómo pueden ayudar a clarificar las semánticas de las notaciones y conceptos orientados a objetos [24]. Ejemplos de dicho trabajo son: el modelo de objetos esencial del OMG que hace uso de Z [25], la formalización de la notación OOA mediante Z y los métodos Syntropy [19] y Fusion [20].

La integración de las técnicas formales, dentro de los métodos orientados a objetos, es parte de un área general de trabajo conocida como integración de métodos [25]. El uso industrial de los métodos formales con frecuencia se alcanza gracias a dicha integración. En [27] se indica que el 31% de aquellas compañías que aplicaron métodos formales lo hicieron en conjunto con métodos diagramáticos semiformales. Este criterio tiene ventajas decisivas en términos de la minimización de las interrupciones a las prácticas de desarrollo existentes.

Tanto Fusion como Syntropy, son parte de una tendencia hacia un mayor grado de abstracción en la especificación de objetos, fuera de los conceptos de métodos y paso de mensajes.

En la integración se han seguido dos enfoques básicos:

- Uso de notación matemática para mejorar las representaciones diagramáticas, teniendo en cuenta, que el desarrollo se realiza predominatemente con uso de las últimas.

- Uso de un proceso de traducción, a partir de los modelos diagramáticos de análisis hacia notaciones formales, tales como un lenguaje de especificación orientado a objetos. El desarrollo se logra fundamentalmente usando las notaciones formales.

El primer enfoque es más adecuado para un "riguroso" aunque no completo, desarrollo orientado a objetos, dado que las notaciones diagramáticas, no soportan una prueba o una verificación. El segundo es más adecuado cuando se desarrollan sistemas altamente integrados.

Como ejemplos del primer enfoque pueden citarse a Fusion y Syntropy, descritos más adelante. En [28] se encuentra un ejemplo del segundo enfoque, que hace uso de B AMN como el lenguaje formal. En [24], se encuentra otro ejemplo que hace uso de este enfoque, recurriendo a LOTOS.

Fusion se generó a partir del trabajo de Coleman, Dollin y otros en Hewlett Packard Labs sobre la mejora a notaciones formales orientadas a objetos tales como los diagramas de estado [20]. Se proveen notaciones semiformales (inglés estructurado) para precondiciones, postcondiciones e invariantes de operaciones y se hace uso de los conceptos básicos de especificaciones formales (abstracción, refinamiento mediante el modelo de exclusión, especificación de operaciones haciendo uso de precondiciones y postcondiciones), sin adoptar notación matemática.

Syntropy es un método más reciente en la misma dirección, que combina los diagramas de estado y las notaciones del modelo de objetos de OMT [29], con los diagramas de interacción y permite el uso de las notaciones Z sobre estos diagramas, para especificar precondiciones y postcondiciones, invariantes y restricciones. Una característica importante de Syntropy es su énfasis en eventos más que en mensajes, para la modelación de los requerimientos del sistema. Un evento puede involucrar cualquier cantidad de objetos (por ejemplo, puede disparar una transición de los diagramas de estado de varias clases de objetos). Esta sincronización (no dirigida) entre objetos más tarde puede implementarse como el paso dirigido de mensajes (llamada de métodos). Esto es análogo a la sincronización, mediante acciones en el Object Calculus, descrito en [30], y a diferencia de Fusion, se da un tratamiento al problema de concurrencia. Syntropy, nació después de muchos años de experiencia en consultoría y ha sido utilizado en aplicaciones financieras críticas en el Reino Unido.

Tanto Fusion como Syntropy, son parte de una tendencia hacia un mayor grado de abstracción en la especificación de objetos, fuera de los conceptos de métodos y paso de mensajes. Ambos enfoques usan conceptos (operaciones del sistema en Fusion, eventos en Syntropy) en la etapa de análisis, que permiten una especificación flexible de los elementos de un sistema, sin un compromiso prematuro con una arquitectura particular de objetos o de definiciones dentro de clases particulares.

La herramienta Venus para OMT y VDM++, provee soporte para el uso combinado de técnicas formales y técnicas digramáticas orientadas a objetos, de manera altamente integrada [31] y está diseñada como una extensión de una herrmienta CASE, ampliamente utilizada para OMT, permitiendo a los usuarios de técnicas orientadas a objetos, incrementar la formalidad de sus desarrollos de manera gradual. Es posible producir una especificación OMT del conjunto de clases, atributos y asociaciones en un sistema, generar un esquema de clases VDM++ a partir de la especificación, mejorar estas clases y entonces traducir de regreso la especificación mejorada dentro de una vista OMT. La

Las técnicas formales permiten un significado preciso para mecanismos complejos orientados a objetos tales como la agregación o los diagramas de estado.

ventaja de este enfoque, es que las herramientas de análisis y edición más apropiadas, pueden usarse sobre las diferentes vistas (diagramáticas vs formales) del sistema. Por lo general, es más efectivo crear y modificar una jerarquía de clases, usando un editor de diagramas que a través de una herramienta basada en texto.

La traducción de notaciones diagramáticas a notaciones formales, tales como las realizadas por Venus, proveen una interpretación particular y semánticas propias a las primeras. Entonces al concepto no bien definido de "agregación" en orientación a objetos, se le puede dar un significado preciso, mediante la traducción a un lenguaje formal como LOTOS [24], VDM++ o Z++. En VDM++, la interpretación se hace a través del concepto de "herencia indexada".

Similarmente, las ambigüedades de la notación de diagramas de estado, puede resolverse mediante una interpretación formal tal como la que se propone en [32] para RTL. Por ejemplo, los métodos de Booch [33] y OMT, toman diferentes interpretaciones considerando si los eventos son encolados, ignorados o si arriban en el momento cuando el objeto destino no está en un estado, en el cual puedan ser aceptados.

TECNOLOGÍAS ORIENTADAS A OBJETOS EN MÉTODOS FORMALES

Las limitaciones de Z y VDM, para la especificación de grandes sistemas de manera modular, condujeron a diferentes investigaciones con el propósito de añadir mecanismos de estructuración a estos lenguajes. Por ejemplo, en [34] se hizo la propuesta de adicionar un mecanismo de "capítulos" para descomponer las especificaciones Z dentro de módulos, y [35] identificaron un enfoque similar, usando una combinación de HOOD y Z. Abrial y otros en BP Research, desarrollaron una extensión basada en objetos de Z, conocida como la notación de la Máquina Abstracta B [36], que ha tenido sus principales aplicaciones en sistemas de transporte seguros [37]. El lenguaje SmallVDM [28], combinó una estructuración basada en objetos derivada de Smalltalk con la notación VDM. Otros trabajos que proveen unas extensiones mínimas a Z y que permiten un estilo de especificación orientado a objetos, incluyen a [38].

Los investigadores y seguidores de las tecnologías de objetos, se encaminaron en la dirección de mecanismos orientados a objetos, porque éstos parecen complementar de modo natural los lenguajes de especificación basados en modelos [39] y [28], apoyándose en las siguientes razones:

- La orientación a objetos estimula la creación de abstracciones y las técnicas formales proveen los mecanismos para describir de modo preciso dichas abstracciones.

- La orientación a objetos provee los mecanismos de estructuración y las disciplinas de desarrollo requeridas para escalar hacia técnicas formales en grandes sistemas.

- Las técnicas formales permiten un significado preciso para mecanismos complejos orientados a objetos tales como la agregación o los diagramas de estado.

La primera extensión totalmente orientada a objetos de Z fue el lenguaje Object-Z [17] y caracteriza los siguientes aspectos claves, de la estructuración orientada a objetos:

- Encapsulación de datos y operaciones dentro de módulos que además definen tipos.

- La posibilidad de creación de subclases de clases, por medio del mecanismo de herencia y la habilidad para usar operaciones polimórficamente, entre una subclase y sus superclases.

- La habilidad para usar instancias de clases dentro de otras clases (composición de clases).

No obstante esto, inicialmente se restringió la composición, de modo que fuese acíclica (la

clase A podría no contener una instancia de una clase B, si B contuviera una instancia de A u otras situaciones cíclicas). Se encontró que la orientación a objetos provee ventajas en términos de convergencia con los conceptos de dominio.

Otras extensiones a Z tales como Z++[40], MooZ [41], OOZE [21] y ZEST [39], se desarrollaron en los años 1989 a 1991. De estos, OOZE se distingue como el único lenguaje que se basa en un enfoque de especificación algebraica, que hace uso de OBJ y FOOPS como su fundamento pero con una sintaxis similar a Z. En el mundo de VDM, el lenguaje Fresco se desarrolló como un medio de proveer un soporte de especificación y verificación formal para el desarrollo con Smalltalk [42].

Varios de estos lenguajes se generaron de modo particular en los campos de las telecomunicaciones y del control de procesos. El trabajo de BT haciendo uso de ZEST es notable a este respecto, y fue tan exitoso que ZEST ahora se utiliza con preferencia a lenguajes de especificación más tradicionales en telecomunicaciones tales como LOTOS y SDL [43]. Los conceptos de identidad de objetos y de estructuras de composición cíclicas fueron introducidos en Z++ y Object-Z [44].

El lenguaje VDM++ representa un enfoque significativamente más ambicioso con base en las ideas de VDM, Smalltalk y la extensión DRAGOON de Ada [45]. Incluye concurrencia y características de sincronización al estilo Ada, trazas (como en CSP) y aspectos de tiempo real tomados de las ideas de Hayes [46]. Como en el caso de Fusion y ZEST, la participación de una compañía importante (en este caso CAP Gemini) ha provisto rutas directas para aplicaciones industriales significativas [47]. Los aspectos de concurrencia y tiempo real, se contemplan dentro de Z++, de una manera más declarativa gracias al uso de lógica de tiempo real [48]. La investigación de extensiones similares para Object-Z continúa. Aspectos destacados en el tratamiento de tiempo real y concurrencia, son la transformación de modelos continuos a discretos de un sistema [47] y la definición de refinamiento y subtipificación en la presencia de concurrencia y comportamiento de tiempo real.

Z no es ideal como base para un lenguaje de especificación orientado a objetos debido a su complejidad semántica, particularmente en el cálculo de esquemas. Algunos intentos para mezclar el cálculo de esquemas y la llamada a métodos, han conducido a una complejidad sintáctica excesiva, como en el caso de Z++ y Object-Z. Ambos lenguajes cambiaron sustancialmente la sintaxis y la interpretación de los esquemas de Z, con el propósito de lograr un formalismo adecuado.

En ciertos casos los problemas de combinar la orientación a objetos y la especificación formal, se deben al lenguaje de programación que instrumenta varios de los conceptos de la orientación a objetos. La invocación de métodos en particular, se interpreta de una manera procedimental y la tarea de encontrar una abstracción adecuada de la llamada a métodos o de usarlos de una manera declarativa (como predicados sobre el proveedor de pre y pos estados) aún no está enteramente resuelta. Debido a que VDM contiene un sublenguaje procedimental, resulta más simple proveer una combinación de especificación formal y orientación a objetos a este nivel. Los mecanismos de modularidad de VDM son aún más primitivos que los de Z, y han sido remplazados completamente por los mecanismos de orientación a objetos en VDM++.

Una desventaja significativa con el uso de estructuración orientada a objetos, es el grado al cual esto complica el razonamiento acerca de las especificaciones

Una desventaja significativa con el uso de estructuración orientada a objetos, es el grado al cual esto complica el razonamiento acerca de las especificaciones, por las siguientes razones:

- El polimorfismo y el enlace dinámico, hacen difícil el control de la versión de un método, que será realmente ejecutada, en una invocación particular [49].

- La herencia, puede ser usada para fragmentar la definición de un método a través de mu-

chas clases, haciendo su significado difícil de determinar [50].

Además, algunos lenguajes orientados a objetos estimulan el uso amplio de los alias y de redes interconectadas de objetos [51].

Hasta hace poco, no existían semánticas formales para un lenguaje de especificación orientado a objetos. Ahora, esto ha sido parcialmente remediado con un sistema de razonamiento y semánticas axiomáticas provistas por Object-Z [52] y Z++ [40] y unas semánticas de notación para MooZ [53].

FUNDAMENTOS DE LOS MÉTODOS FORMALES

Varias áreas de las matemáticas proporcionan el andamiaje formal para la tecnología de objetos: la lógica, la teoría de tipos, la teoría de categorías [15] y el álgebra de procesos [54]. La lógica provee una forma de describir y razonar acerca del comportamiento de los objetos y de las clases a un nivel que es razonablemente transparente y entendible. La teoría de categorías provee un marco matemático general en el cual varias clases de estructura, pueden ser descritas de modo natural. Mediante el desarrollo de estructuras de teorías lógicas, se ha probado que es posible proveer una caracterización concisa de diversas estructuras que se usan en tecnologías orientadas a objetos.

Un objeto puede ser considerado como de diferentes subtipos de una clase en diferentes momentos. Las clases se modelan como conjuntos de entidades de objetos.

El álgebra de procesos provee una forma alternativa de caracterización, mediante la consideración de las clases y los objetos como procesos y mediante el uso del álgebra para combinarlos de diversas maneras. El cálculo-π, es particularmente relevante para expresar las semánticas orientadas a objetos, por que su uso del paso de nombres entre procesos, es análogo a la reconfiguración dinámica en sistemas de objetos.
El enfoque lógico se ejemplifica en [30]. Las especificaciones de lógica temporal de primer orden, definen las propiedades de los objetos, incluyendo los efectos, las restricciones de permisos y los requerimientos de comportamiento de los métodos, que se interpretan como símbolos de acción en la lógica, mientras que los atributos se interpretan como símbolos de atributos (que son distintos de las variables en el sentido lógico, dado que no pueden ser cuantificados y tienen valores dependientes del tiempo).

Los modelos teóricos de conjuntos de la orientación a objetos [56], proveen un medio para discutir la identidad de los objetos y aspectos tales como la migración de subtipos [57], donde un objeto puede ser considerado como de diferentes subtipos de una clase en diferentes momentos. Las clases se modelan como conjuntos de entidades de objetos. Una definición clara de subtipificación y refinamiento no emerge necesariamente de tales nociones debido a problemas que involucran clases definidas recursivamente.

La teoría de tipos ha sido estudiada extensivamente como base para las semánticas de la orientación a objetos. En [58] y [59] se dan ejemplos del enfoque teórico de tipos. Este enfoque interpreta clases como tipos (tales como tipos recursivos y polimórficos o como tipos cuantificados existencialmente), y provee respuestas definitivas a los problemas de subtipificación. El trabajo en esta área, incluye el cálculo de objetos [60], en el que por analogía con el uso del cálculo-λ para definir las semánticas de los lenguajes funcionales, las semánticas de la orientación a objetos se expresan en un cálculo basado en los conceptos de método, objeto y tipo como elementos primitivos. Este formalismo puede usarse para darle un significado a las extensiones del paradigma orientado a objetos, por ejemplo "métodos como objetos".

Un enfoque lógico, donde las clases se interpretan como teorías, provee una definición aparentemente natural de subtipificación y refinamiento como extensión teórica. Así, la clase D es una subclase de la clase C, si satisface todas los requerimientos de C, a través de la traducción sintáctica. Esta definición cumple con los principios de subtipificación informales de [61].

En un lenguaje de especificación, las clases de objetos representan entidades conceptuales y de diseño, mientras que los procesos son una entidad más relacionada a la implementación del sistema.

Sin embargo, para propósitos prácticos, la extensión teórica no puede ser usada de por sí, en cambio, se usa un conjunto de pruebas que generalmente son suficientes pero no necesarias, para la suptipificación o refinamiento a sostener. Estas pruebas son descompuestas sobre la base de entidades de clase, y son similares a las de [62].

Las distinciones entre subtipificación y refinamiento han sido clarificadas recientemente. En algunos enfoques de modelación, estos conceptos han sido combinados, mientras que en otros, tal como en el álgebra de procesos de LOTOS, con frecuencia no es claro qué definición de "simulación de procesos" debería usarse como la base de verificación [63]. Similarmente, la determinación del concepto correcto de bisimulación para el cálculo-π aún es un área activa de investigación [64].

El refinamiento puede verse como una forma de subtipificación (es decir, extensión teórica), pero con restricciones adicionales. En particular, no está permitido introducir nuevos métodos externos en el refinamiento de clases, aunque nuevos métodos internos puedan ser definidos. El concepto de adecuación de VDM (es decir, para todo estado abstracto existe un estado concreto bajo la interpretación datos de clases y atributos [4]), también se requiere para refinamiento pero no para subtipificación.

CONCURRENCIA

Es conocido que los objetos, debido a su independencia y al aislamiento de su estado interno, de la interferencia por otros objetos, son unidades apropiadas de ejecución concurrente. Como ejemplos de algunos intentos por integrar los conceptos de concurrencia y la orientación a objetos pueden citarse la notación POOL [65], DRAGOON [66] y VDM++.

Sin embargo, el ajuste entre objetos y procesos no es exacto. En particular, los objetos pueden ser concurrentes internamente, si representan un sistema que debe responder más rápidamente a la entrada de datos que a la velocidad que puede alcanzarse por vía secuencial. Además, varios objetos pueden estar sobre el mismo procesador y entonces pueden considerarse como parte del mismo proceso. En un lenguaje de especificación, las clases de objetos representan entidades conceptuales y de diseño, mientras que los procesos son una entidad más relacionada a la implementación del sistema.

CONCLUSIONES

La combinación de técnicas formales y de las tecnologías orientadas a objetos, ha alcanzado avances significativos en ambos campos y ha conducido a una mayor diseminación del trabajo en métodos formales, que de otra manera podría no haber ocurrido. Por ejemplo, las presentaciones sobre el cálculo de objetos, han tenido un lugar preeminente en las conferencias OOSPLA, TOOLS y ECOOP.

Los retos futuros para las técnicas formales en tecnologías de objetos incluyen un tratamiento formal de patrones [67] y marcos (arquitecturas específicas de dominio para sistemas de objetos en ese dominio). Además, las herramientas de soporte para métodos formales constituyen un área de gran interés para la futura adopción a escala industrial de estos métodos, en pos de la mejor calidad del software.

REFERENCIAS

[01] Pérez C. J. N. *Línea de investigación en administración de redes*. Universidad Distrital Francisco José de Caldas, Facultad de Ingeniería, Maestría en Teleinformática, Junio de 1992.

[02] Pérez C. J. N. *Sistema Integrado de Pruebas, Análisis y Gestión de Redes (SIPAG)*. Universidad Distrital Francisco José de Caldas, Facultad de Ingeniería, Maestría en Teleinformática, Noviembre de 1994.

[03] Spivey M. *The Z notation: A reference manual*. Prentice Hall, 2nd edition, 1992.

[04] Jones C. *Systematic software development using VDM*. Prentice Hall International. (2nd edition), edition 1992.

[05] Ritchie B. *Proof with Mural*. Rutherford Appleton Laboratory, Informatics Department, Chilton, Didcot, Oxon OX11 0QX, UK, 1993.

[06] Andrews D. J. *Information Techonology Programming Langauges - VDM-SL; First Commitee Draft Standard: CD 13817-1*, documento iso/iec jtcl/sc22/wg19 n-20, Noviembre de 1993.

[07] Ehrig H. y Mahr B. *Fundamentals of algebraic specification2*. Springer-Verlag 1990.

[08] Guttag J. *Abstract data types and the development of data structures.* En Programming Language Design. Los Alamitos, CA: IEEE Computer Society Press, 1980
[09] Goguen J. y Ginali S. *A categorical approach to general systems theory.* En G. Klir, Editor, Applied General Systems Research, págs. 257-270. Plenum, 1978.
[10] Hoare, C. *Communicating sequential processes.* Prentice Hall, 1985.
[11] Milner R. *Communication and concurrency.* Prentice Hall, 1991.
[12] Milner R. *The polyadic* p*-calculus: A tutorial.* En M. Broy, editor, Logic and Algebra of Specification, 1992.
[13] May D. *Use of formal methods by silicon manufacturer.* En C. A. R. Hoare, editor, Developments in Concurrency and Communication, capítulo 4, págs. 107-129. Addison Wesley, 1990.
[14] Brinksma E. *Information Processing Systems - Open Systems Interconnection - LOTOS - A formal descripción technique based on the temporal ordering of observation, ISO 8807*, 1988.
[15] Goguen J. y Diaconescu R. *Towards an algebraic semantics for the object paradigm.* En Harmut Ehrig and Fernando Orejas Editors, Recent Trends in Data Type Specification. Srpinger-Verlag Lecture Notes in Computer Science, 785, 1994.
[16] George C. et al. *The RAISE specification language.* Prentice Hall, 1992.
[17] Carrington D., et al. *Object-Z: An object oriented extension to Z,* En Formal Description Techniques, II (FORTE'89), págs 281-196. North-Holland, 1990.
[18] Durr E. y Katwinjk J. *VDM++: a formal specification language for object-oriented design.* En TOOLS Europe'92 págs: 63-77, 1992.
[19] Cook S. y Daniels J. *Designing Object Systems: object-oriented modelling with Syntropy.* Prentice Hall, 1994.
[20] Coleman D. et al. *Object oriented development: The Fusion method.* Prentice Hall - object-oriented series, 1994.
[21] Alencar A. J. y Goguen J. A. *OOZE: An object-oriented Z environment.* En P. America, editor, ECOOP'91 Proceedings, volumen 512 de Lecture Notes in Computer Science, págs. 180-199. Springer-Verlag. Julio de 1991.
[22] Hoare C. *Communicating sequential processes.* Prentice Hall, 1985.
[23] Jones C. *An object-based design method for concurrent programs.* Technical Reports UMCS-92-12-1, Department of Computer Science, University of Manchester, 1992.
[24] Moreira A. M. D. y Clark R. G. *Combining object-oriented analysis and formal description techniques.* En M. Tokoro and R. Pareschi, editors, 8th European Conference on Object-Oriented Programming: ECCOP'94, LNCS 821, págs. 344-364. Springer-Verlag, Julio de 1994.
[25] Houston I. *Formal specification of the OMG Core Object Model.* Technical Report IBM UK, Hursely Park, 1994.
[26] Semmens L. et al. *Integrated structured analysis and formal specification techniques.* The Computer Journal, 35(6), 1992.
[27] Austin S. y Parkin G. I. *Formal methods: a survey. Technical report,* National Physical Laboratory, Queens Road, Tedington, Middlesex, TW11 OLW, Marzo de 1993.
[28] Lano K. y Haughton H. *Object-oriented specification case studies*. Prentice Hall, first edition, 1993.
[29] Rumbaugh J. et al. *Object-oriented modelling and design.* Prentice-Hall, 1991.
[30] Fiadeiro J. y Maibaum P. *Towards object calculi*. Technical Report, Imperial College, 1991.
[31] LeBlanc P. *VENUS User manual combined use of OMT and VDM++.* Technical Report afro/verilog/plb/um/v2,3, Verilog, 1995.
[32] Barroca L. M. *et al. The architectural specification of an avionic subsystem.* En IEEE Workshop on industrial-trength formal specification techniques, págs. 17-29. IEEE Press, 1995.
[33] Booch G. y Bryan D. *Software Engineering inb Ada*, cuarta edición. Benjamin/Cummings, 1994.
[34] Sampaio A. y Meria S. *Modular extensions to Z.* En VDM and Z, volume 428 of Lecture Notes in Computer Science. Springer-Verlag, 1990.
[35] Iachini P. y Giovanni R. *HOOD and Z for the development of complex software systems*. En VDM and Z VDM 90, volumen 428 de Lecture Notes in Computer Science, págs. 262-289. Springer-Verlag, 1990.
[36] Haugton H. y Lano K. *B Abstract machine notation: a reference manual.* Mc Graw Hill, 1995.
[37] Bowen J. y Stavridou V. *Safety-critical systems, formal methods and standars.* Software Engineering 1993.
[38] Hall A. *Specificying and interpreting class hierarchies in Z.* En 8th Z User Meeting, Workshop in Computing, Springer-Verlag, 1994.
[39] Cusack E. *Object-oriented modelling in Z.* En P. America, editor, ECOOP'91 Proceedings, Lecture Notes in Computer Science. Springer Verlag, 1991.
[40] Lano K. *Z++, an object oriented extension to Z.* In J. Nicholls, editor, Z User Meeting, Oxford, UK Workshops in Computing. Springer-Verlag, 1991.
[41] Meira S. R. L. y Cavalcanti A. L. C. *Modular object-oriented Z specifications.* En Z User Meeting 1990, Workshops in Computing, págs 173-192. Springer-Verlag, 1991.
[42] Wills A. *Capsules and types in Fresco: Program verification in Smalltalk*. En P. America, editor, ECCOP'91 Proceedings, volume 512 of Lecture Notes in Computer Science, págs. 59-76. Springer-Verlag, 1991.
[43] Moller-Pedersen B et al. *Relational and tutorial on osdl: An object-oriented extension of sdl.* Computer Networks and ISDN Systems, 13(2): 97-117, 1987.
[44] Lano K. *Refinement in object-oriented specification languages.* En D. Till. Editor, 6th Refinement Worshop. Springer-Verlag, 1994.
[45] Atkinson W. D. *et al*, *Modal action logic for the specification and validation of safety*. En Mathematical Structures for Software Engineering. The Institute of Mathemtics and its Applications Conference Series 27, Clarendon Press, 1991.
[46] Hayes J. y Mahony B. *A case study in timed refinement: A mine pump.* IEEE Software, 18(9), Septiembre de 1992.
[47] Durr E. y Dusink E. *The role of VDM++ in the development of a real-time tracking and tracing system.* En J. Woodcock and P. Larsen, Editors, FME'93 Lecture Notes in Computer Science. Springer Verlag, 1993.
[48] Lano K. *Reactive system specification and refinement.* En TAPSOFT'95, Volume 915 of Lecture Notes in Computer Science. Springer-Verlag, 1995.
[49] Ponder C. y Bush B. *Polymorphism considered harmful.* ACM Sigplan Notices, 27(6), Junio de 1992.
[50] Wide N. y Huit R. *Maintenance support for object-oriented programs.* En Proceedings of Conference on Software Maintenance. IEEE Computer Society Press, 1991.
[51] Hogg J. *Islands: Aliasing protection in object-oriented languages.* En OOSPLA'91 Proceedings. Springer-Verlag, 1991.
[52] Smith G. *A logic for object-Z*. En Z User Meeting'95. Lecture Notes in Computer Science. Springer-Verlag, 1995.
[53] Lin T. *A formal semantics for MooZ*, PhD Thesis. Technical Reports, DI/UFPE, Recife/PE, Brazil, 1994.
[54] Hennesy M. *Algebraic theory of processes.* The MIT Press, 1988.
[55] Malcom G. y Goguen J. *Proving correctness of refinement and implementation.* Technical monograph PRG-114, Programming Resarch Group, Oxford, University, 1994.
[56] Maung I. et al. *Towards a formatization of programming by difference.* En FME'94 Proceedings. Springer-Verlag, 1994.
[57] Wieringa R. et at. *Roles and dynamic subclasses: a modal logic approach*. En ECCOP'94 Proceedings. Springer-Verlag, 1994]
[58] Cook W. y Palsberg J. A. *A denotational semantics of inheritance and its correctness.* Proceedings of International Confeence on Object-oriented programming, systems and languages, 24(10): 433-443. Special Issue of SIGPLAN Noteces, 1989.
[59] Hense A. *Denotational semantics of an object-oriented language with explicit wrapper.* BCS Formals Aspects of Computing, 5:181-207, 1993.
[60] Abadi M. y Cardelli L. *An imperative object calculus*. En P.D. Mosses, M. Nielsen, y M. I. editores, TAPSOFT'95, volumen 915 de Lecture Notes in Computer Science. Springer-Verlag, Mayo de 1995.
[61] Liskov B. *Data abstraction and hierarchy.* En OOSPLA'87 (Addendum to proceedings): ACM SIGPLAN Notices, 23(5): 17-34, Mayo de 1988.
[62] Liskov B. y Wing J. *Family values: A behavioral notion of subtyping.* Technical Report CMU-CS-93-187, School of Computer Science Carnegie Mellon University, 1993.
[63] Thomas M. y Kirkwood C. *Experiences with specification and verification in LOTOS*. En Industrial-Strength Formal Specification Techniques. IEEE Press, 1995.
[64] Lin T. *Complete inference systems for weak bisimulation equivalences in the* p*-calculus.* En TAPSOFT'95, volume 915 of Lecture Notes in Computer Science. Springer-Verlag, Mayo de 1995.
[65] America P. *Pool-t: a parallel object oriented language.* En Object Oriented Concurrente Programming. MIT Press, 1987.
[66] Atkinson C. *Object Oriented Reuse, Concurrency and Distribution*, ACM Press: Addison Wesley, 1991.
[67] Goldberg A. y Robson D. *Smalltalk-80: The language and its implementation.* Addison Wesley, 1983.

José Nelson Pérez Castillo

Profesor Universidad Distrital Francisco José de Caldas
Investigador Centro de las Telecomunicaciones CINTEL.
Ingeniero de sitemas y Magister en Teleinformática de la UD.
Especialista en Sistemas de Información Geográfica, Teledetección y Cartografía, Universidad de Alcalá de Henares, España
Doctor en Informática, Universidad de Oviedo, España.
e-mail: jonepe@cc-net.net