Ingeniare. Revista Chilena de Ingeniería
ISSN: 0718-3291
facing@uta.cl
Universidad de Tarapacá
Chile

Serna M., Edgar; Serna A., Alexei
La especificación formal en contexto: actual y futuro
Ingeniare. Revista Chilena de Ingeniería, vol. 22, núm. 2, abril, 2014, pp. 243-256
Universidad de Tarapacá
Arica, Chile

La especificación formal en contexto: actual y futuro

Formal specification in context: current and future

Edgar Serna M.[1] Alexei Serna A.[2]

Recibido 31 de enero de 2013, aceptado 23 de octubre de 2013
Received: January 31, 2013 Accepted: October 23, 2013

RESUMEN

La especificación formal es un área de investigación activa en la ingeniería de *software* de este siglo, en la que se aplica en diversas configuraciones y técnicas, y aunque su uso industrial todavía es limitado, la comunidad científica tiene actualmente una comprensión diferente acerca de su utilidad y necesidad. Hasta el momento el trabajo de los investigadores se focaliza en la especificación escrita durante el diseño del modelo funcional preliminar, por lo que se centra principalmente en evaluar las herramientas relacionadas. En este trabajo se realiza una revisión a la literatura, se hace un recorrido por la esencia, la función, el uso y los inconvenientes de las técnicas de especificación formal y se analizan algunos criterios de valoración y de evaluación a sus debilidades. Los resultados se convierten en la base para formular trabajos futuros, con el objetivo de buscar que la especificación formal se afiance como actividad básica de investigación.

Palabras clave: Técnicas formales, especificación formal, métodos formales, ciencias computacionales, ingeniería de *software*.

ABSTRACT

Formal specification is an active research field in software engineering of this century, where different configurations and techniques are employed and although their industrial use is still limited, the scientific community currently has a different understanding about its usefulness and necessity. Until present time, researchers' work focuses on the specification written during the design of the preliminary functional model, for this reason it mainly focuses on evaluating the related tools. In this paper, a literature review is performed, covering the essence, function, use and disadvantages of formal specification techniques and some assessment and evaluation criteria are analyzed regarding their weaknesses. The results become the basis for future work formulation, aimed to the strengthening of formal specification as a core research activity.

Keywords: Formal techniques, formal specification, formal methods, computer sciences, software engineering.

INTRODUCCIÓN

Desde el surgimiento de las Ciencias Computacionales los investigadores han considerado a la especificación formal (FS por sus siglas en inglés) como una de sus áreas de interés. Finalizando los años cuarenta Turing [1] observó que el razonamiento acerca de programas secuenciales era más sencillo cuando, en puntos específicos del mismo, se hacían anotaciones acerca de las propiedades de su estado. En los años

[1] Facultad de Ingenierías. Ingeniería de Sistemas. Instituto Tecnológico Metropolitano. Calle 54 Nº 30-01. Medellín, Colombia. E-mail: edgarserna@itm.edu.co
[2] Grupo de Investigación CCIS. Instituto Antioqueño de Investigación. Medellín, Colombia. E-mail: alexei.serna@fundacioniai.org

sesenta Floyd [2], Hoare [3] y Naur [4] propusieron técnicas axiomáticas para demostrar la consistencia entre los programas secuenciales y esas propiedades, a las que llamaron *especificaciones*, y Dijkstra [5] demostró cómo utilizar constructivamente el cálculo para derivar programas no determinísticos que las cumplieran. Parnas [6] y Liskov [7] propusieron técnicas específicas para expresar formalmente las propiedades de los programas, particularmente para datos estructurados, y Pnueli [8] lo hizo para programas concurrentes. Estos aportes conformaron el punto de partida para la *especificación formal* como una nueva área de investigación [9, 11], y desde entonces se ha incrementado continuamente el interés que despertó, lo mismo que los múltiples usos en la ingeniería de software [12-14]. Hasta el momento el objetivo principal de esta comunidad de investigadores se focaliza en la especificación escrita durante el diseño del modelo funcional preliminar [12], por lo que su investigación, alrededor de la cual se genera este trabajo, se centra principalmente en evaluar y comparar las herramientas relacionadas con esta especificación y en analizar sus fortalezas y debilidades.

En este artículo se presenta el resultado de una revisión a la literatura para determinar la esencia, la función, el uso y los inconvenientes de las técnicas de especificación formal. Se discuten algunos criterios de valoración y se hace una evaluación a sus fortalezas y debilidades. Los resultados se convierten en la base para formular una serie de trabajos futuros, con el objetivo de que la especificación formal se afiance como una actividad básica de investigación en la ingeniería de *software* de este siglo.

METODOLOGÍA

El procedimiento para realizar esta investigación se orientó a identificar estudios y opiniones que pudieran ser candidatos a incluir o excluir del conjunto final de la revisión. Las fuentes fueron los buscadores en internet, las bases de datos digitales de periódicos, revistas, asociaciones industriales y académicas y en las bibliotecas. Los parámetros de búsqueda incluyeron las palabras: *formal specification* y *formal languages*, combinadas con *techniques*, *metodologies*, *methods*, *research*, *industrial application*, *requirments engineering* y *software engineering*. Estas combinaciones debían aparecer en el título o en el contenido del documento. Para lograr mayor cubrimiento no se determinó una línea de tiempo específica y no se excluyó ninguna fuente inicial. Además, en los estudios primarios seleccionados se validó la solidez de la metodología aplicada y de los resultados presentados, y las opiniones y/o análisis las debían presentar instituciones o personas que estuvieran relacionadas con el área de investigación, y que su trayectoria fortaleciera el postulado.

En la muestra final se incluyeron directamente: 1) los artículos de revistas en bases de datos, por haber pasado por procesos de selección y evaluación estructurados, 2) los documentos publicados en los *blogs*, cuyos autores y empresas fueran idóneos en el área y 3) los artículos en periódicos y revistas, los cuales debían haber escrito autores con trabajo en FS y experiencia en los campos relacionados. Al final, la muestra inicial quedó conformada por 154 trabajos. Además de estas características, debían hacer, en su totalidad, un aporte relevante a la temática de investigación. Para alcanzar los objetivos se aplicaron las siguientes fases:

1. Identificar los estudios relevantes.
2. Excluir estudios basados en el título.
3. Excluir estudios a base de resúmenes.
4. Analizar los estudios y seleccionar los más relevantes para la temática en función del texto completo.

Los criterios de inclusión y exclusión más importantes fueron: formalidad y pertinencia del sitio donde se aloja, autoridad del o de los autores, calidad y aportes del contenido, fuentes de datos, sustentación de la tesis, calidad de la investigación y coherencia de los resultados. Luego de este proceso se excluyeron 38 trabajos de la muestra inicial y con los 116 restantes se realizó el análisis que se presenta a continuación.

RESULTADOS

La Especificación Formal

El término Especificación Formal se puede referir a diversos aspectos en el ciclo de vida del *software* y se utiliza indistintamente para un producto como para su correspondiente proceso, por lo que está sobrecargado en la literatura. Generalmente se acepta que una especificación formal es la expresión, en algún lenguaje formal y con cierto nivel de abstracción, de una serie de características que el

sistema debe satisfacer. Esta definición se utiliza dependiendo de lo que se comprenda como *sistema*, del tipo de *características* a satisfacer, del nivel de *abstracción* aplicado y del *lenguaje formal* utilizado [15]. El sistema puede ser un modelo descriptivo del dominio de las necesidades, del *software* y su entorno, del *software* como tal, de la interfaz de usuario, de la arquitectura del *software* o de algún proceso a seguir, entre otros. Las características se pueden referir a los objetivos de alto nivel, a los requisitos no funcionales, a la hipótesis del dominio, o a los protocolos de interacción entre esos componentes. Pero, para asegurar que una solución resuelve correctamente un problema y más allá de las diferentes concepciones de especificación, existe una idea común relacionada con el dominio del problema: *se debe establecer el primer estado en el que este es correcto*. Sin embargo, esta dicotomía es simplista porque generalmente una solución se puede representar como un conjunto de sub-problemas, los cuales se especifican y resuelven a la vez [16]; por lo tanto, una especificación debe cumplir con alguna característica de alto nivel y también satisfacer algunas de bajo nivel.

Otra cuestión que se determinó en la revisión es que, frecuentemente, el término *formal* se confunde con *precisión* y, aunque la incluye, lo contrario no es cierto. El lenguaje para expresar una especificación formal se expresa con: 1) reglas de *sintaxis*, para determinar oraciones gramaticalmente bien formadas, 2) reglas *semánticas*, para interpretar oraciones de forma precisa y significativa en el dominio y 3) reglas de *inferencia* para deducir información útil desde la especificación (teoría de las pruebas), que constituyen la base para automatizar el análisis. Normalmente el conjunto de características que se especifica es grande, por lo que el lenguaje debe permitir que se organicen en unidades vinculadas mediante estructuras de relación, como la especialización, la agregación, la instanciación, el enriquecimiento, el uso, entre otras. Cada una de estas unidades tendrá: 1) una parte declarativa, donde se declaran las variables de interés y 2) una parte de aserción, donde se formalizan las propiedades del objeto en las variables declaradas.

Por otro lado, escribir una especificación *correcta* es difícil, probablemente tanto como escribir un programa *correcto.* Se puede considerar *buena* si cumple con: 1) ser *adecuada*, indicar adecuadamente el estado del problema en cuestión, 2) tener *consistencia interna,* poseer una interpretación semántica significativa que haga verdadero a todo el conjunto de propiedades especificadas, 3) no ser *ambigua*, tener una sola interpretación de lo que es cierto, 4) ser *completa*, respecto del nivel superior donde las propiedades especificadas sean suficientes para establecerla [17], 5) ser *satisfecha*, por el nivel inferior y 6) ser *mínima*, no puede tener propiedades de estado irrelevantes para el problema o que solo lo sean para la solución del mismo [18].

Ventajas de la formalización

La formalización es un proceso necesario para diseñar, validar, documentar, comunicar, hacer reingeniería y reutilizar soluciones informáticas. Además, permite obtener especificaciones con mayor nivel de calidad y proporciona las bases para su automatización. Otras aplicaciones las describen para plantear preguntas y para detectar problemas serios en la formulación informal, y utilizan la semántica del formalismo para establecer normas precisas de interpretación que permitan superar varios problemas del lenguaje natural. Las ventajas encontradas fueron:

- Derivar permisos o consecuencias lógicas de la especificación para confirmar usuarios por medio del teorema deductivo de las técnicas de prueba [18].
- Confirmar que una especificación operacional satisface especificaciones más abstractas, o generar contraejemplos de comportamiento por el modelo algorítmico de las técnicas de comprobación [21-23].
- Generar contraejemplos de las afirmaciones de una especificación declarativa [24].
- Generar escenarios concretos que ilustren características deseadas o no deseadas de la especificación [25], o inferir inductivamente la especificación de escenarios [26].
- Producir animaciones de la especificación para comprobar si es adecuada [27-28].
- Comprobar eficientemente las formas específicas de consistencia/completitud de la especificación [29].
- Generar excepciones de alto nivel y pre-condiciones de conflicto que puedan hacer que la especificación no se cumpla [30-31].
- Generar especificaciones de alto nivel, como invariantes o condiciones de fortaleza [32, 34].

- Transferir refinamientos de la especificación para generar obligaciones de prueba [35-36].
- Generar casos de prueba y oráculos desde la especificación [38-39].
- Soportar la reutilización formal de los componentes mediante la especificación correspondiente [40-41].
- Ser la base para la ingeniería inversa y para la evolución del *software* desde el código [42].

Principios de la Especificación Formal

Como productos de los procesos investigativos y de la experiencia del trabajo de los investigadores en la revisión se encontraron los siguientes principios:

- *No es formal*. Primero se debe determinar qué son las propiedades de estado para poderlas precisar y formalizar, por lo que es necesario formularlas en un lenguaje que las partes interesadas puedan utilizar y comprender: el lenguaje natural.

- *No tiene sentido si no cuenta con una definición informal precisa acerca de cómo interpretarla en el dominio*. Una formalización implica términos y predicados que pueden tener varios y diferentes significados, por lo que tendrán sentido si se definen con precisión mediante la asignación de nombres a las funciones/predicados y a las funciones/relaciones entre los objetos del dominio. Esta asignación debe ser precisa e informal para evitar la regresión infinita, un principio que a menudo se pasa por alto [43].

- *Es más que un simple proceso de traducción de lo informal a lo formal*. La especificación de un sistema complejo requiere objetos relevantes y fenómenos que se identifican, relacionan y caracterizan mediante propiedades compartidas. La construcción del modelo y la descripción de las propiedades están íntimamente unidos a los componentes de cualquier proceso de especificación.

- *Es difícil de desarrollar y evaluar*. Esto se debe a la diversidad y sofisticación de los errores y a la multiplicidad de opciones de modelado que se puedan tener. Como consecuencia, la especificación formal raramente es *correcta* al primer instante; sin embargo, también se señala que incluso la especificación *incorrecta* puede ayudar a encontrar problemas en la formulación original.

- *La razón para especificar las opciones de modelado es que es importante para su explicación y evolución*. Una justificación que raramente se documenta [44].

- *Los subproductos de la especificación formal frecuentemente son más importantes que ella misma.* Porque incluyen una especificación más informal que obtienen de la retroalimentación, estructuración y análisis de la expresión formal y de otros productos de menor nivel.

- *Para que un sistema formal sea útil debe tener un dominio de aplicabilidad limitado*. Los tipos específicos de sistemas requieren tipos específicos de técnicas para expresarlos naturalmente y para analizarlos eficientemente.

La Especificación Formal en contexto

Cada año se incrementa el número de casos de éxito al utilizar especificaciones formales en sistemas reales, lo que se refleja en trabajos de reingeniería [11] y de desarrollo de sistemas [78-79]. En este último caso se reportan evidencias de que su aplicación no incrementa los costos de desarrollo, mientras que se mejora la calidad de los productos.

Aunque la mayoría de estudios se aplica en los sistemas de transporte, también se reportan casos de éxito en los sistemas de información, los sistemas de telecomunicaciones, el control de plantas de energía, los protocolos y la seguridad. Algunos ejemplos se pueden encontrar en [11, 13-14]. Se destaca el trabajo de Behm [79], quien describe lo sucedido en el sistema del metro de París cuando se abrió una nueva línea con tráfico totalmente controlado por *software* y con trenes sin conductor. Los componentes de seguridad crítica de a bordo, a lo largo de la pista y en el piso, se desarrollaron formalmente utilizando el método de máquina abstracta de B [56]. Ese desarrollo incluyó modelos abstractos de los componentes, el refinamiento para modelos concretos y una traducción automatizada a código ADA. De acuerdo con el autor, el sistema posee cerca de 100.000 líneas de especificación en B, que cubren los modelos abstracto y concreto, y 87.000 líneas de código ADA; el refinamiento se

validó mediante pruebas formales; la herramienta B generó automáticamente 28.000 lemas y el 65% de las normas fueron añadidas a las pruebas de descarga. De esta forma fue posible encontrar la mayoría de errores, que se fijaron posteriormente en el desarrollo concurrente. Además, luego de desplegar el proceso convencional de pruebas no se encontraron errores diferentes.

El éxito en este estudio de caso se podría explicar porque: 1) el lenguaje B tiene una base matemática simple, que les permite a los ingenieros utilizarlo después de un período razonablemente corto de formación, 2) la técnica de especificación multi-nivel permite pasar relativamente fácil desde el modelo abstracto al código y de manera apropiada y demostrable, 3) el soporte metodológico se presentó en forma de directrices y heurísticas, con el objetivo de guiar los procesos de desarrollo y de validación, 4) se diseñó un modelo explícitamente para el proceso de desarrollo/validación, que se integró al de la empresa para adaptarlo a prácticas convencionales como las pruebas (la falta de integración se reconoce como un obstáculo importante para adoptar los métodos formales [74]) y 5) el proceso se soportó en herramientas robustas. En el siglo pasado, la madurez tecnológica de las herramientas de especificación se incrementó notablemente y su eficacia logró el análisis de especificaciones formales y la derivación de información útil. El desempeño en especificaciones grandes también se incrementó y cada vez fueron más fáciles de utilizar (los animadores de especificación y los modelos de prueba fueron particularmente útiles a este respecto). Por otro lado, se originó una tendencia prometedora hacia la integración múltiple, con el objetivo de ofrecer un amplio espectro de análisis a diversos costos (el conjunto de herramientas SCR es un ejemplo de esta tendencia [73]).

Técnicas de Especificación Formal

Las técnicas de especificación formal reportadas en la literatura se detallan en la Tabla 1.

EVALUACIÓN Y COMPARACIÓN

Para realizar la evaluación y comparación de las técnicas halladas en la revisión se definieron varios

Tabla 1. Técnicas descritas en los trabajos primarios.

Técnica	Características
Basadas en historias	Describen la mayor cantidad de historias (comportamientos) en el tiempo. Las propiedades se especifican de acuerdo con las aserciones de la lógica temporal en los objetos del sistema, mediante operadores relacionados con los estados pasados, presentes y futuros, que pueden ser lineales [8] o ramificados [45]. Estas estructuras de tiempo pueden ser discretas [46-47], densas [48], o continuas [49], y las propiedades se pueden referir a puntos de tiempo [46-47], intervalos de tiempo [50], o a ambos [51]; frecuentemente se especifican en las fronteras del tiempo, sin embargo, las lógicas temporales son necesarias en tiempo real [52-53].
Basadas en estados	Describen el sistema sobre la base de los estados arbitrarios ocurridos en un tiempo. Las propiedades se especifican con invariantes que restringen los objetos del sistema, como una fotografía instantánea, o por pre y postaserciones, que restringen su aplicación operacional en cualquier instante. Esto puede ser explícito o implícito, dependiendo de si la aserción contiene o no ecuaciones que definan constructivamente la salida. Lenguajes como Z [10, 54-55], VDM [56] y B [36] aplican estas técnicas. También se han propuesto las variantes orientadas por objetos [57].
Basadas en transiciones	Describen las transiciones requeridas desde un estado a otro. Las propiedades se especifican mediante un conjunto de funciones de transición en la máquina de estados, que proporciona el estado de salida correspondiente para cada estado de entrada y para cada activación del evento. La ocurrencia de un evento inicial es una condición suficiente para que tenga lugar la transición correspondiente (a diferencia de una precondición, captura una obligación), y las poscondiciones se especifican para proteger la transición. Lenguajes como Statecharts [58], PROMELA [86], STeP-SPL [20], RSML [31] y SCR [73] se basan en estas técnicas.
Funcionales	Especifican el sistema como un conjunto estructurado de funciones matemáticas: 1) *Algebraica*, donde las funciones se agrupan en los tipos de objetos que aparecen en el dominio o co-dominio para definir las estructuras algebraicas, y las propiedades se especifican como ecuaciones condicionales que capturan el efecto de las funciones; lenguajes como OBJ [60], ASL [61], PLUSS [62] y LARCH [63] se basan en ellas, 2) *Funciones de orden superior*, donde las funciones se agrupan en teorías lógicas que contienen definiciones tipo (predicados lógicos), declaraciones de variables y axiomas, que definen las diversas funciones; lenguajes como HOL [64] y PVS [19, 65] se basan en este enfoque.
Operacionales	Caracterizan el sistema como una colección estructurada de procesos que puede ejecutar una máquina más o menos abstracta. Lenguajes como Paisley [66], GIST [67] y las redes Petri [68] se basan en estas técnicas.

criterios basados en las mismas recomendaciones de los estudios de la muestra final y en la experiencia del trabajo de los autores al respecto. Como era de esperar, algunos son interdependientes e incluso conflictivos, por lo que una elección razonable depende de las prioridades del sistema y de la tarea en cuestión.

Detalle de la especificación y capacidad del código
Cada una de las técnicas analizadas incorpora alguna tendencia semántica, como las *Funcionales,* que se basan en estados, se centran en los comportamientos secuenciales y proporcionan estructuras adecuadas para definir objetos complejos, por lo que tienen mayor aceptación para sistemas transaccionales, o las *Basadas en historias*, en *transiciones* y las *operacionales,* que se enfocan en comportamientos concurrentes y proporcionan solo estructuras simples para definir los objetos a manipular, por lo que su aceptación es mayor para sistemas reactivos. Pero existen enfoques híbridos que intentan ampliar su aplicación, como los propuestos en [69-70]. Más allá de esta tendencia semántica, los lenguajes formales deben permitir que las propiedades del sistema se expresen sin necesidad de utilizar demasiado código ni que la especificación se refiera a la definición de problemas y no a las soluciones de programación. Idealmente, debería existir una correspondencia simple y directa entre la formulación de una propiedad en lenguaje natural y su contraparte formal, pero raramente se da el caso.

A diferencia del lenguaje natural, los formales tienen limitaciones, por ejemplo, en un lenguaje de primer orden es complicado hacer referencia a operaciones como argumentos de predicados, de modo que se necesitan algunos trucos de codificación para superar el problema, como la introducción de eventos auxiliares. Algunos lenguajes formales son débiles para soportar las referencias temporales, las de tiempo, las explícitas o las implícitas, que frecuentemente ocurren en las formulaciones naturales; por ejemplo, cuando se incorporan inhabilidades en las especificaciones basadas en estados para referirse al pasado, por lo que cuando tal o cual evento ocurre es necesario introducir variables auxiliares en la codificación, con las correspondientes operaciones de actualización que se deben especificar en cada modificación del estado, como en la programación imperativa. Las técnicas *basadas en historias* son la principal excepción a este problema, sin embargo, también presentan inconvenientes para especificar ordenaciones relativas a eventos; por ejemplo, Dwyer, Avrunin y Corbett [71] describen un ejemplo, relativamente simple, de una propiedad de pedido que requiere ¡*seis niveles de operador de anidación en lógica temporal lineal*! Las especificaciones algebraicas figuran entre las que requieren más experiencia en codificación, por lo que, debido a los problemas de la expresividad del lenguaje, la codificación de la especificación puede requerir mayor conocimiento especializado.

Capacidad de construcción y nivel de operación
Las técnicas de especificación deberían proporcionar facilidades para construir incrementalmente especificaciones complejas en pequeñas partes, y los cambios locales en las características del problema se deberían reflejar en cambios locales en la especificación de requisitos. Estos requisitos dependen de: 1) mecanismos del lenguaje, para estructurar la especificación y el razonamiento composicional y 2) la disponibilidad de un método incremental, para construir, analizar y modificar. Algunos lenguajes formales soportan mecanismos de estructuración básica para especificaciones modulares, como encapsulamiento, herencia, inclusión, generalización, enriquecimiento, entre otras. Otros lenguajes también soportan relaciones de refinamiento como base para desarrollar y analizar la especificación incremental, por ejemplo, reificación de datos [36, 56], composición/descomposición de componentes mediante conectores lógicos [54], composición/descomposición de estados [58], o abstracción/refinamiento de objetivos [59].

Facilidad de uso
Para los ingenieros de *software* con experiencia y un adecuado nivel de formación debería ser posible escribir especificaciones de buena calidad. En la revisión se encontró que este criterio depende de todos los anteriores más algunos otros. El lenguaje debería tener una base teórica sencilla, lo que probablemente explica la popularidad de los basados en nociones matemáticas simples y bien comprendidas, como conjuntos y relaciones y funciones [10, 19, 36, 54]. Además, el lenguaje también debería eximir complejidades, como la necesidad de especificaciones basadas en estados, por medio de axiomas de marco adicional [72].

Nivel de Comunicación
Contrario al lenguaje, la técnica debería ser accesible para personas razonablemente bien entrenadas, de tal manera que puedan leer y comprobar especificaciones de alta calidad. Este criterio también depende de los anteriores, particularmente de la cercanía entre la especificación y su correspondiente formulación en lenguaje natural, y del formato exterior que la especificación pueda adoptar. Esto explica la popularidad que tienen actualmente las técnicas que soportan formatos tabulares [11, 59, 65] y notaciones diagramáticas [58].

Calidad del Análisis
La efectividad de una técnica depende del grado en que satisfaga los objetivos planteados, porque no tiene sentido escribir especificaciones formales sino van a recibir retroalimentación de las herramientas automatizadas. Este proceso debería soportar un amplio rango de análisis en el espacio de las posibilidades, pero con algunas excepciones notables como la descrita en [73], hasta ahora esencialmente es una ilusión. Por lo general, favorecer a uno u otro tipo de análisis permite la selección de una u otra técnica de especificación. Usualmente, cuanto más eficiente sea el análisis, más esfuerzo de codificación requiere del lado de la especificación.

Este es el caso de la especificación *basada en animaciones* para ejecutar especificaciones operacionales, o para la reescritura de los términos de las algebraicas. Ilustrar el modelo a los probadores está bien, porque el lector no convencido puede ver lo que refleja su código de entrada para una aplicación compleja. Por otro lado, el análisis más completo es la intervención más experta usualmente requerida, y los asistentes de las pruebas son un buen ejemplo de este hecho [74]. Debería ser claro poder traducir cualquier multicriterio de análisis en favor de un marco multiparadigmático, en el que se integraran de forma coherente los formalismos complementarios, los métodos y las herramientas, a fin de combinar lo mejor de cada paradigma para dominios, tareas y problemas específicos. Intentos preliminares se encaminan en esta dirección [75-77].

ANÁLISIS DE RESULTADOS

Deficiencias
A pesar de que los resultados demuestran un progreso constante en el desarrollo de estas técnicas, también es cierto que presentan una serie de deficiencias, algunas de las cuales explican el porqué algunos las consideran no adecuadas para la fase crítica de la especificación y el análisis de requisitos en los sistemas complejos de hoy.

Alcance. La mayoría se limita a la especificación de requisitos funcionales (lo que se espera que haga el sistema) y, generalmente, dejan por fuera de cualquier tratamiento formal a los no funcionales. Una excepción son las técnicas que permiten propiedades de distribución para formalizar y calcular requisitos [37].

Separación de cometidos. La mayoría no proporciona soporte para hacer una separación clara entre los requisitos previstos del sistema, los supuestos acerca del entorno y las propiedades del dominio de la aplicación, por lo que no se puede hacer una diferenciación entre los descriptivos y los propositivos (también llamados *indicativos* y *optativos* [43]), porque se mezclan todos en la especificación.

Ontologías de bajo nivel. En el desarrollo de las soluciones, los conceptos en términos de qué problemas tienen que ser estructurados y formalizados se tratan a nivel de programación (con mayor frecuencia datos y operaciones), pero las exigencias actuales obligan a elevar el nivel de abstracción y la riqueza conceptual encontrada en los documentos de requisitos informales, como los objetivos y sus refinamientos, los agentes y sus responsabilidades, las alternativas, y así sucesivamente [80-81].

Integración. Con algunas excepciones, estas técnicas están aisladas de otros productos y procesos del *software*. Esto se debe a que presentan: 1) *aislamiento vertical*, al no prestar atención a los productos que continúan en la siguiente fase del ciclo de vida del *software* y que dependen de la especificación formal, como objetivos, requisitos y presunciones, ni a los productos que quedaron atrás y que condujeron a la especificación formal, como los componentes arquitecturales y 2) *aislamiento horizontal*, al no prestar atención a los productos que las acompañan y a los que debería estar vinculada la especificación formal, como la especificación informal correspondiente, la documentación de opciones, la validación de datos, la información de gestión de proyectos, entre otras.

Orientación. La literatura de la especificación formal se dedica principalmente a describir un conjunto de notaciones y un análisis *a posteriori* de especificaciones escritas usando esas notaciones y, generalmente, no trata los métodos constructivos para diseñar de forma segura las especificaciones *correctas* para sistemas complejos, sea sistemática o incrementalmente. En lugar de proponer cada vez más lenguajes, la comunidad debería dedicar mayor esfuerzo a elaborar y validar métodos para diseñar y modificar buenas especificaciones.

Nivel intelectual. Generalmente, las técnicas de especificación formal requieren que el ingeniero de *software* posea buena experiencia en sistemas formales (lógica matemática en particular), en técnicas de análisis y en el uso de herramientas de caja blanca y, debido a la escasez de tales profesionales y al alto costo de tenerlos, actualmente su utilización todavía es limitada en los proyectos industriales, a pesar de los beneficios que prometen.

Retroalimentación. La mayoría de técnicas y herramientas de análisis formal son efectivas para señalar los problemas de los sistemas, pero, en general, son deficientes para sugerir las causas de dichos problemas y para proponer acciones de recuperación.

EL FUTURO DE LA ESPECIFICACIÓN FORMAL

Las técnicas y herramientas para la Especificación Formal que se desarrollan actualmente y que se propondrán en el futuro, deberán incorporar los siguientes requisitos y retos para que la especificación formal logre sus objetivos y para que los productos de la ingeniería de *software* logren el nivel de calidad que se espera de ellos.

Enfoque constructivo

El actual enfoque, dedicado casi exclusivamente al análisis *a posteriori* de especificaciones posiblemente pobres, se debe reemplazar por uno más constructivo, en el que la especificación se diseñe de forma incremental desde un nivel más alto. De esta forma se podrá garantizar un adecuado nivel de calidad desde la construcción del producto. Se podría hablar entonces de un método diseñado a partir de una colección de estrategias de construcción de modelos, de reglas de selección de estilos, de reglas para derivar especificaciones y de directrices y heurísticas; unas podrían ser de dominios independientes y otras de dominios específicos. Dicho método deberá proporcionar una activa orientación en los procesos de toma de decisiones, y podría estar soportado por herramientas de especificación automatizadas, que presten asistencia en los puntos de decisión y que registren los procesos subsiguientes para su documentación (y posible reproducción) en caso de una evolución posterior.

Soporte para análisis comparativo

Hasta el momento la investigación en especificación formal revela que diferentes especificadores, aunque posean el mismo conocimiento del sistema, pueden construir especificaciones diferentes para la formulación inicial de una solución. Esto también es cierto para el programa terminado, pero en este caso por lo menos existe un momento de verdad único: *funciona o no funciona satisfactoriamente*. Más allá de esas cualidades para la especificación se necesitan criterios precisos y medidas de evaluación para comparar sus méritos relativos.

Integración

La tecnología del futuro se deberá preocupar por integrar las especificaciones, vertical y horizontalmente, al ciclo de vida del *software*; partiendo desde los objetivos de alto nivel para el diseño funcional de componentes arquitectónicos y desde la formulación informal hasta la especificación formal de los productos relacionados.

Nivel de abstracción

Las técnicas deben pasar del diseño funcional a la ingeniería de requisitos, donde el impacto de los errores es aún más crucial. Se requiere lenguajes, métodos y herramientas que soporten ontologías más robustas y que se orienten a los problemas superiores, y programas que permitan soportarlas correctamente. Los intentos preliminares en este sentido incluyen al refinamiento orientado por objetivos [37, 82], el análisis de conflicto a nivel de objetivos [30, 37] y el manejo de excepciones a nivel de objetivos [31].

Mecanismos estructurados

La mayoría de las propuestas disponibles hasta el momento para modularizar especificaciones grandes se han copiado desde sus homólogos de la programación. Las propuestas orientadas a

problemas deberían estar disponibles como, por ejemplo, puntos de vista de las partes interesadas o de las visiones del problema [83].

Alcance
Las técnicas de especificación necesitan extenderse para hacer frente a las diferentes categorías de requisitos no funcionales, como rendimiento, integridad, confidencialidad, exactitud de la información, disponibilidad, tolerancia a fallos, costos operativos, mantenibilidad, entre otros, que se elicitan en la ingeniería de requisitos y que desempeñan un destacado papel en el diseño arquitectónico. Las técnicas de razonamiento cualitativo son un primer paso en esta dirección [82]. Categorías específicas pueden requerir lenguajes y técnicas de análisis específicos.

Nivel de complejidad
El uso de especificaciones formales no debe requerir conocimientos profundos en sistemas formales; la complejidad matemática debe estar oculta y las herramientas de análisis se deben poder utilizar como compiladores. El trabajo en especificación basada en patrones [71] es un paso muy prometedor en esta dirección, además, se puede aplicar para reutilizar pruebas y generar especificaciones [31, 37].

Especificación múltiple
Los sistemas complejos tienen múltiples facetas y debido a que un paradigma aislado no es útil para todos los propósitos, debido a sus sesgos semánticos, se necesitan *frameworks* en los que se puedan combinar múltiples paradigmas de forma semánticamente válida, de tal forma que se exploten las mejores características de cada uno. Las diversas facetas necesitan estar vinculadas mediante reglas de consistencia. Los *frameworks* multiparadigmáticos deben ser capaces de integrar a diversos lenguajes formales, los semiformales y el natural, junto con las correspondientes técnicas y herramientas de análisis. Los intentos lingüísticos preliminares en esta dirección combinan redes semánticas, especificación basada en historias y estados, o basada en estados y en transiciones [77]. Mientras que la integración multilingüe es relativamente fácil de alcanzar con los lenguajes semiformales, los formales plantean problemas semánticos difíciles.

Multianálisis
Un *framework* multiparadigmático debe soportar, opcionalmente, diferentes niveles de análisis, desde los simples como el de nivel superficial (análisis de trazabilidad, controles estáticos de semántica y razonamiento cualitativo), hasta los más complejos, como el análisis a nivel profundo (verificación de algoritmos, razonamiento deductivo, o razonamiento inductivo a partir de ejemplos). Las opciones más complejas se deben utilizar solo cuando se requieran y cuando sean necesarias. En las primeras etapas, un entorno multianálisis también les debe permitir a los usuarios utilizar las instalaciones típicas de las herramientas CASE estándar, y llegar a fases más complejas de los métodos formales a medida que logren confianza en su aplicación.

Aplicación temprana
Muchas técnicas requieren que la especificación esté completa para poder realizar el análisis necesario. En el futuro se necesitará que el análisis comience mucho antes, en la etapa temprana de la ingeniería de requisitos [62, 85] y que se incremente posteriormente. Esto garantizará la recuperación anticipada de la inversión e incrementará las utilidades (un importante objetivo señalado en [14]). Por otro lado, las técnicas deductivas también asumen que la especificación es consistente para obtener información útil, que se deriva especialmente en el contexto de la ingeniería de requisitos, donde se puede inferir desde puntos de vista conflictivos. Para derivar tal información se necesitan sistemas formales y técnicas de razonamiento, a pesar de las inconsistencias temporales [84].

Reutilización
Es probable que los problemas en el dominio se asemejen a las soluciones, por lo que la reutilización de la especificación deberá ser más prometedora que la reutilización del código. Sorprendentemente, las técnicas para recuperar, adaptar y consolidar especificaciones reciben relativamente poca atención en la literatura. La reutilización de especificaciones que resulten ser buenas y eficaces para sistemas similares se deberá abordar con enfoques constructivos para la especificación del futuro.

Métricas
Para que la especificación formal logre mejores y mayores objetivos en el futuro, deberá ser posible medir los beneficios de utilizarla en la ingeniería de *software* mediante métricas similares a las utilizadas para medir el incremento de la productividad del

software. Esta cuestión recibe poca atención en los procesos investigativos, de acuerdo con la revisión realizada.

CONCLUSIONES

El *software* se ha convertido en un producto tecnológico imprescindible para la actual Sociedad de la Información y el Conocimiento. Cada vez más se necesitan productos *software* de calidad y las especificaciones formales ofrecen un amplio espectro de posibles caminos para alcanzar ese objetivo. Por este y otros aspectos, la FS recibe actualmente una atención creciente en el mundo académico e industrial. Sin embargo, existe un extenso camino por recorrer antes que se puedan utilizar completamente en los procesos de la ingeniería de *software*. Entre los retos que plantea su utilización se requieren otros factores críticos para lograr el éxito que se espera de ella en el futuro:

- La prestación de asistencia constructiva en el desarrollo de la especificación, el análisis y la evolución.
- La integración vertical y horizontal de las especificaciones formales dentro del ciclo de vida del *software*.
- Abstracciones de alto nivel para la especificación y el análisis de requisitos.
- La disponibilidad de técnicas formales para requisitos no funcionales.
- Las interfaces livianas para la especificación y el análisis multiparadigmático.
- En lugar de solo señalar los problemas, las herramientas del futuro deberán ayudar a resolverlos.

Actualmente, los métodos formales son una realidad y la industria y la comunidad de investigadores continúan desarrollando técnicas de especificación formal que en el futuro funcionarán adecuadamente y ofrecerán las ventajas que se espera de ellas, incluso para aplicaciones críticas. Desde las técnicas formales han surgido buenas prácticas de especificación y desarrollo de *software* y no se pueden dejar de lado ni suplantar, el objetivo es mejorarlas. Los futuros trabajos deberán reunir los principios de la automatización total de las pruebas, determinar las formas en que la formalidad se puede utilizar con mayor efectividad y buscar que los procesos formativos la incluyan como parte de los contenidos curriculares en ciencias computacionales. Los desarrolladores y la industria deben comenzar a ver a los métodos formales y por ende a la especificación formal como una inversión y trabajar para garantizar que el producto logre la calidad que el usuario espera por su valor.

REFERENCIAS

[1] B. Randell. "The Origin of Digital Computers". Springer-Verlag. Berlin. 1973.

[2] R. Floyd. "Assigning Meanings to Programs". Mathematical Aspects of Computer Science. Vol. 19, pp. 19-32. 1967.

[3] C.A.R. Hoare. "An Axiomatic Basis for Computer Programming". Communications of the ACM. Vol. 12, Issue 10, pp. 576-583. 1969.

[4] P. Naur. "Proofs of algorithms by General Snapshots". BIT. Vol. 6, pp. 310-316. 1969.

[5] E.W. Dijkstra. "Guarded commands, non-determinacy and the formal derivation of programs". Communications of the ACM. Vol. 18, Issue 8, pp. 453-457. 1975.

[6] D.L. Parnas. "A Technique for Software Module Specification with Examples". Communications of the ACM. Vol. 15, Issue 5, pp. 330-336. 1972.

[7] B.H. Liskov and S.N. Zilles. "Specification Techniques for Data Abstractions". IEEE Transactions on Software Engineering. Vol. 1, Issue 1, pp. 7-18. 1975.

[8] A. Pnueli. "The Temporal Logics of Programs". 18th IEEE Symposium on Foundations of Computer Science FOCS, pp. 46-57. Rhode Island, USA. 31 Oct.-1 Nov., 1977.

[9] D.L. Parnas. "The Use of Precise Specifications in the Development of Software". IFIP Congress 77, pp. 849-867. Toronto, Canadá. Aug. 8-12, 1977.

[10] J.R. Abrial. "The Specification Language Z. Syntax and Semantics". Oxford University Press. USA. 1980.

[11] K.L. Heninger. "Specifying Software Requirements for Complex Systems: New Techniques and their Application". IEEE Transactions on Software Engineering. Vol. 6, Issue 1, pp. 2-13. 1980.

[12] J.M. Wing. "A Specifier's Introduction to Formal Methods". IEEE Computer. Vol. 23, Issue 9, pp. 8-24. 1990.

[13] M.G. Hinchey and J.P. Bowen. "Applications of Formal Methods". Prentice-Hall. USA. 1995.

[14] E.M. Clarke and J.M. Wing. "Formal Methods: State of the Art and Future Directions". ACM Computing Surveys. Vol. 28, Issue 4, pp. 626-643. 1996.

[15] E. Serna M. "Formal Methods and Software Engineering". Rev. Virtual Universidad Católica del Norte. N° 30, pp. 158-184. 2010.

[16] W. Swartout and R. Balzer. "On the Inevitable Intertwining of Specification and Implementation". Communications of the ACM. Vol. 25, Issue 7, pp. 438-440. 1982.

[17] K. Yue. "What Does It Mean to Say that a Specification is Complete?". Fourth International Workshop on Software Specification and Design, pp. 34-41. Monterey, California, USA. April 3-4, 1987.

[18] B. Meyer. "On Formalism in Specifications". IEEE Software. Vol. 2, Issue 1, pp. 6-26. 1985.

[19] S. Owre; J. Rushby and N. Shankar. "Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS". IEEE Transactions on Software Engineering. Vol. 21, Issue 2, pp. 107-125. 1995.

[20] Z. Manna and the STep Group. "STeP: Deductive-Algorithmic Verification of Reactive and Real-Time Systems". Lecture Notes in Computer Science. Vol. 1102, pp. 415-418. 1996.

[21] E.M. Clarke and E.A. Emerson. "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications". ACM Transactions on Programming Languages and Systems. Vol. 8, Issue 2, pp. 244-263. 1986.

[22] J.M. Atlee. "State-Based Model Checking of Event-Driven System Requirements". IEEE Transactions on Software Engineering. Vol. 19, Issue 1, pp. 24-40. 1993.

[23] C. Heitmeyer, J. Kirby, B. Labaw, M. Archer and R. Bharadwaj. "Using Abstraction and Model Checking to Detect Safety Violations in Requirements Specifications. IEEE Transactions on Software Engineering. Vol. 24, Issue 11, pp. 927-948. 1998.

[24] D. Jackson and C.A. Damon. "Elements of Style: Analyzing a Software Design Feature with a Counterexample Detector". IEEE Transactions on Software Engineering. Vol. 22, Issue 7, pp. 484-495. 1996.

[25] R.J. Hall. "Explanation-Based Scenario Generation for Reactive System Models". Automated Software Engineering. Vol. 7, Issue 2, pp. 157-177. 2000.

[26] A. Lamsweerde and L. Willemet. "Inferring Declarative Requirements Specifications from Operational Scenarios". IEEE Transactions on Software Engineering. Special Issue on Scenario Management, pp.1089-1114. 1998.

[27] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-trauring and M. Trakhtenbrot. "STATEMATE: A Working Environment for the Development of Complex Reactive Systems". IEEE Transactions on Software Engineering. Vol. 16, Issue 4, pp. 403-414. 1990.

[28] J.M. Thompson, M.E. Heimdahl and S.P. Miller. "Specification-Based Prototyping for Embedded Systems". ESEC/FSE'99, pp. 163-179. Toulouse, France. September 6, 1999.

[29] M.P. Heimdahl and N.G. Leveson. "Completeness and Consistency in Hierarchical State-Based Requirements". IEEE Transactions on Software Engineering. Vol. 22, Issue 6, pp. 363-377. 1996.

[30] A. Lamsweerde, R. Darimont and E. Letier. "Managing Conflicts in Goal-Driven Requirements Engineering". IEEE Transactions on Software Engineering. Vol. 24, Issue 11, pp. 908-926. 1998.

[31] A. Lamsweerde and E. Letier. "Handling Obstacles in Goal-Oriented Requirements Engineering". IEEE Transactions on Software Engineering. Vol. 26, Issue 10, pp. 978-1005. 2000.

[32] A. Lamsweerde and M. Sintzoff. "Formal Derivation of Strongly Correct Concurrent Programs". Acta Informatica. Vol. 12, Issue 1, pp. 1-31. 1979.

[33] D.Y. Park, J. Skakkebaek and D.L. Dill. "Static Analysis to Identify Invariants in RSML Specifications". 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, pp. 133-142. Lyngby, Denmark. September 14-18, 1998.

[34] R. Jeffords and C. Heitmeyer. "Automatic Generation of State Invariants from Requirements Specifications". ACM SIGSOFT Software Engineering Notes. Vol. 23, Issue 6, pp. 56-59. 1998.

[35] C. Morgan. "Programming from Specifications". USA, Prentice-Hall. 1990.

[36] J.R. Abrial. "The B-Book: Assigning Programs to Meanings". Cambridge University Press. USA. 1996.

[37] R. Darimont and A. Lamsweerde. "Formal Refinement Patterns for Goal-Driven Requirements Elaboration". ACM SIGSOFT Software Engineering Notes. Vol. 21, Issue 6, pp. 179-190. 1996.

[38] G. Bernot, M.C. Gaudel and B. Marre. "Software Testing Based on Formal Specifications: A Theory and a Tool". Software Engineering Journal. Vol. 6, Issue 6, pp. 387-405. 1991.

[39] D.J. Richardson, A.S. Leif and T.O. O'malley. "Specification-based test oracles for reactive systems". International Conference on Software Engineering, pp. 105-118. Melbourne, Australia. May 11-15, 1992.

[40] H.B. Reubenstein and R.C. Waters. "The Requirements Apprentice: Automated Assistance for Requirements Acquisition". IEEE Transactions on Software Engineering. Vol. 17, Issue 3, pp. 226-240. 1991.

[41] P. Massonet and A. Lamsweerde. "Analogical Reuse of Requirements Frameworks". Third IEEE International Symposium on Requirements Engineering, pp. 26-37. Annapolis, MD, USA. January 5-8, 1997.

[42] G.C. Gannod and B.H. Cheng. "Strongest Postcondition Semantics as the Formal Basis for Reverse Engineering". Journal of Automated Software Engineering. Vol. 3, pp. 139-164. 1996.

[43] P. Zave and M. Jackson. "Four Dark Corners of Requirements Engineering". ACM Transactions on Software Engineering and Methodology. Vol. 6, Issue 1, pp. 1-30. 1997.

[44] J. Souquières and N. Levy. "Description of Specification Developments". First IEEE Symposium on Requirements Engineering, pp. 216-223. San Diego, California, USA. January 12-14, 1993.

[45] E.A. Emerson and J.Y. Halpern. "Sometimes and not Never Revisited: on Branching versus Linear Time Temporal Logic". Journal of the ACM. Vol. 33, Issue 1, pp. 151-178. 1986.

[46] Z. Manna and A. Pnueli. "The Temporal Logic of Reactive and Concurrent Systems". UK, Springer-Verlag. 1992.

[47] L. Lamport. "The Temporal Logic of Actions". ACM Transactions on Programming Languages and Systems. Vol. 16, Issue 3, pp. 872-923. 1994.

[48] S.J. Greenspan, A. Borgida and J.A. Mylopoulos. "A Requirements Modeling Language and its Logic". Information Systems. Vol. 11, Issue 1, pp. 9-23. 1986.

[49] K.M. Hansen, A.P. Ravn and H. Rischel. "Specifying and Verifying Requirements of Real-Time Systems". IEEE Transactions on Software Engineering. Vol. 19, Issue 1, pp. 41-55, 1993.

[50] L. Moser, Y.S. Ramakrishna, G. Kutty, P. M. Melliar-Smith and L.K. Dillon. "A Graphical Environment for the Design of Concurrent Real-Time Systems". ACM Transactions on Software Engineering and Methodology. Vol. 6, Issue 1, pp. 31-79. 1997.

[51] F. Jahanian and A.K. Mok. "Safety Analysis of Timing Properties in Real-Time Systems". IEEE Transactions on Software Engineering. Vol. 12, pp. 890-904. 1986.

[52] R. Koymans. Specifying message passing and time-critical systems with temporal logic". Springer-Verlag. USA. 1992.

[53] E. Dubois, J. Hagelstein and A. Rifaut. "A Formal Language for the Requirements Engineering of Computer Systems". In Thayse, A. (Ed.), "Introducing a Logic Based Approach to Artificial Intelligence". John Wiley and Sons Ltd., pp. 357-433. USA. 1991.

[54] J.M. Spivey. "The Z Notation - A Reference Manual". Prentice-Hall. USA. 1992.

[55] B. Potter; J. Sinclair and D. Till. "An Introduction to Formal Specification and Z". Prentice-Hall. USA. 1996.

[56] C.B. Jones. "Systematic Software Development using VDM". Prentice-Hall. USA. 1990.

[57] K. Lano. "Formal Object-Oriented Development". Springer-Verlag. London. 1995.

[58] D. Harel. "Statecharts: A Visual Formalism for Complex Systems". Science of Computer Programming. Vol. 8, pp. 231-274. 1987.

[59] D.L. Parnas and J. Madey. "Functional Documents for Computer Systems". Science of Computer Programming. Vol. 25, Issue 1, pp. 41-61. 1995.

[60] K. Futatsugi, J.A. Goguen, J.P. Jouannaud and J. Meseguer. "Principles of OBJ. ACM Symposium on Principles of Programming Languages POPL'85, pp. 52-66. New Orleans, Louisiana, USA. January 14-16, 1985.

[61] E. Astesiano. "An introduction to ASL". Germany, Universitat Passau, Fakultat fur Mathematik und Informatik. 1986.

[62] M.-C. Gaudel. "Structuring and Modularizing Algebraic Specifications: the PLUSS specification language, evolutions and perspectives". Lecture Notes in Computer Science. Vol. 557, pp. 1-18. 1992.

[63] J.V. Guttag and J.J. Horning. "LARCH: Languages and Tools for Formal Specification". UK, Springer-Verlag. 1993.

[64] M. Gordon and T.F. Melham. "Introduction to HOL". Cambridge University Press. USA. 1993.

[65] J. Crow, S. Owre, J. Rushby, N. Shankar and M. Srivas. "A Tutorial Introduction to PVS". Workshop on Industrial-Strength Formal Specification Techniques WIFT'95. Boca Raton, USA. April. 2-5, 1995.

[66] P. Zave. "An Operational Approach to Requirements Specification for Embedded Systems". IEEE Transactions on Software Engineering. Vol. 8, Issue 3, pp. 250-269. 1982.

[67] R.M. Balzer, N.M. Goldman and D.S. Wile. "Operational Specification as the Basis for Rapid Prototyping". ACM SIGSOFT Software Engineering Notes. Vol. 7, Issue 5, pp. 3-16. 1982.

[68] R. Milner. "Communication and Concurrency". USA, Prentice-Hall. 1989.

[69] S.R. Faulk, J.W. Brackett, P. Ward and J. Kirby. "The CORE Method for Real-Time Requirements". IEEE Software. Vol. 9, Issue 5, pp. 22-33. 1992.

[70] A. George, A. Haxthausen, S. Hughes, R. Milne, S. Prehn and J.A. Pedersen. "The RAISE Development Method". USA, Prentice-Hall. 1995.

[71] M.B. Dwyer, G.S. Avrunin and J.C. Corbett. "Patterns in Property Specifications for Finite-State Verification". 21th Intrnational Conference on Software Enginering ICSE-99, pp. 411-420. Los Angeles, CA, USA. May 16-22, 1999.

[72] A. Borgida, J. Mylopoulos and R. Reiter. "On the Frame Problem in Procedure Specifications". IEEE Transactions on Software Engineering. Vol. 21, Issue 10, pp. 785-798. 1995.

[73] C. Heitmeyer, J. Kirby, B. Labaw and R. Bharadwaj. "SCR*: A Toolset for specifying and Analyzing Software Requirements". 10th Annual Conference on Comuter-Aided Verification CAV'98, pp. 526-531. Vancouver, Canadá. June 28July 2, 1998.

[74] D. Craigen, S. Gerhart and T. Ralston. "Formal Methods Technology Transfer: Impediments and Innovation". In M.G. Hinchey and J.P. Bowen (Eds.), "Applications of Formal Methods". pp. 399-419. Prentice-Hall. USA. 1995.

[75] C. Niskier, T. Maibaum and D. Schwabe. "A Pluralistic Knowledge-Based Approach to Software Specification". 2nd European Software Engineering Conference ESEC-89, pp. 411-423. Coventry, UK. September 11-15, 1989.

[76] P. Zave and M. Jackson. "Conjunction as Composition". ACM Transactions on Software Engineering and Methodology. Vol. 2, Issue 4, pp. 379-411. 1993.

[77] P. Zave and M. Jackson. "Where Do Operations Come From? A Multiparadigm Specification Technique". IEEE Transactions on Software Engineering. Vol. 22, Issue 7, pp. 508-528. 1996.

[78] A. Hall. "Using Formal Methods to Develop an ATC Information System". IEEE Software. Vol. 12, Issue 6, pp. 66-76. 1996.

[79] P. Behm, P. Benoit, A. Faivre and J.M. Meynadier. "Météor: A Successful Application of B in a Large Project". Wold Congress on Formal Methods in the Development of Computing Systems, pp. 369-387. Toulouse, France. September 20-24, 1989.

[80] M. Feather. "Language Support for the Specification and Development of Composite Systems". ACM Transactions on Programming Languages and Systems. Vol. 9, Issue 2, pp. 198-234. 1987.

[81] J. Mylopoulos. "Information Modeling in the Time of the Revolution". Information

Systems. Vol. 23, Issue 3-4, pp. 127-155. 1998.

[82] J. Mylopoulos, L. Chung and B. Nixon. "Representing and Using Nonfunctional Requirements: A Process-Oriented Approach". IEEE Transactions on Software Engineering. Vol. 18, Issue 6, pp. 483-497. 1992.

[83] D. Jackson. "Structuring Z Specifications with Views". ACM Transactions on Software Engineering and Methodology. Vol. 4, Issue 4, pp. 365-389. 1995.

[84] A. Hunter and B. Nuseibeh. "Managing Inconsistent Specifications: Reasoning, Analysis and Action". ACM Transactions on Software Engineering and Methodology. Vol. 7, Issue 4, pp. 335-367. 1998.

[85] E. Serna M. "Engineering Requirements management by a semi-formal model". In Press, 2013.

[86] B. Vlaovic, A. Vreze, Z. Brezocnik and T. Kapus. "Automated Generation of Promela Model from SDL Specification". Computer Standards & Interfaces. Vol. 29, Issue 4, pp. 449-461. May, 2007.