Especificaciones Formales Tempranas del Comportamiento de Sistemas de Software

Fernando Asteasuain -Manuel Dubinsky – Federico Díaz – Juan Lagostena
Contacto: fasteasuain@undav.edu.ar
Ing. en Informática – Dpto. Tecnología y Administración – Universidad Nacional de Avellaneda

Resumen

El objetivo general del presente proyecto es facilitar la especificación formal del comportamiento de artefactos de software. La especificación formal del comportamiento esperado de artefactos de software ha sido identificada como uno de los mayores obstáculos para el desarrollo de software basado en modelos, y para la transferencia de técnicas de validación y verificación formal como model checking.

La comunidad en Ingeniería de Software cree que una posible solución debe poder combinar varios condimentos: deben permitir la descripción parcial del comportamiento, la facilidad para especificar sistemas abiertos, se deben usar sintaxis simples y conocidas, deben basarse en notaciones con suficiente poder expresivo y deben soportar heterogeneidad (combinando declaratividad para estar cerca de la especificación de los requerimientos y notaciones operacionales basadas en autómatas o formalismos similares).

El objetivo específico de esta investigación es el desarrollo de un nuevo lenguaje declarativo con el suficiente poder expresivo para especificar el comportamiento de sistemas abiertos, y la capacidad para operacionalizar las especificaciones. En el mismo se combinará la posibilidad de describir comportamiento parcial, permitiendo el modelado incremental, junto con la posibilidad de especificar el comportamiento de artefactos describiendo su interacción con un ambiente o contexto externo, habilitando las especificaciones de sistemas abiertos.

Palabras claves: *Lenguaje Declarativo*, *Modelado de Comportamiento*, *Especificación de Comportamiento*.

Contexto

El presente proyecto se encuentra enmarcado dentro del proyecto UNDAVCYT denominado "Especificaciones formales tempranas del comportamiento de sistemas de software", dirigido por el Dr. Fernando Asteasuain. El grupo de investigación está conformado por los autores del presente trabajo. El proyecto tiene una duración de dos años, y actualmente está en ejecución el segundo año del proyecto. El proyecto está financiado 100% por la Universidad Nacional de Avellaneda.

1. Introducción

El desarrollo de software libre de errores representa el mundo ideal para la Ingeniería de Software, y todas las herramientas y técnicas buscan desde hace años acercarse cada vez más a ese objetivo de máxima. En la actualidad se estima que buena parte del tiempo que insume el desarrollo de software se lo lleva el proceso de detección y corrección de errores o "bugs" [8]. Diversas técnicas se han elaborado desde la comunidad de Ingeniería de Software para intentar atacar este problema. Entre ellas, el proceso de Testing ha ido creciendo en importancia y alcance, desde los clásicos tests de caja blanca, negra, de unidad, regresión, hasta métodos más complejos como la generación automática de casos de test [10] o el testing de mutación [11], entre otros. Pese a todo el crecimiento de esta área, el proceso de testing es por naturaleza incompleto, ya que no garantiza la ausencia de errores. La comunidad busco entonces apoyo en áreas como la validación y verificación formal de software, y el desarrollo de software basado en modelos [27]. Entre estas técnicas se destaca Model Checking [1], que en pocas palabras busca

establecer de manera automática si un modelo dado de un sistema satisface o no reglas que describensucomportamiento.

Sin embargo, existen obstáculos para poder transferir estas herramientas al mundo industrial del desarrollo de software [2-4]. Uno de los más importantes es la dificultad para completar el ciclo de la especificación de propiedades y de modelos: los lenguajes formales usados para la especificación requieren usuarios expertos, y las especificaciones obtenidas en ocasiones noson concisas, y resultan en ocasiones difíciles de manipular, entender, comunicar y validar [3, 4, 12]. La especificación de propiedades es un proceso que incluye dos pasos: 1) escribir la propiedad en un lenguaje formal y 2) revisar, validad esa propiedad para asegurarse que está describiendo el comportamiento que se tiene en mente. La mayoría de las aproximaciones (por ejemplo, basadas en lógicas temporales como LTL o notaciones basadas en autómatas) requieren usuarios expertos para poder escribir las propiedades, lo cual representa un importante obstáculo a la hora de adoptar técnicas formales de verificación [2,4,12]. Adicionalmente, estas aproximaciones no proveen al usuario de instrumentos que faciliten la validación de las propiedades: es una tarea compleja revisar propiedades, compararlas, modificarlas. Todos estos son aspectos claves a la hora de explorar y entender las implicancias de una propiedad.

Otro punto mencionado como un problema para los lenguajes de especificación es su poder expresivo. Muchos autores han señalado problema de expresividad en lógicas temporales como LTL [18,21] y se han propuesto extensiones como ser la inclusión de automátas de Büchi en lógicas temporales [21]. Sin embargo, estas extensiones vuelven demasiado complejos los lenguajes, debilitándose su poder de uso.

Uno de los intentos para atacar estos problemas resulta dellenguaje declarativo Feather Weight Visual Scenarios (FVS)[12,17,28]. Uno de los principales aportes de FVS fue facilitar la tarea de la descripción y validación de propiedades, proveyendo un lenguaje gráfico y declarativo, donde se obtienen especificaciones simples y concisas. Además, FVS cuenta con un alto poder expresivo, capaz incluso de expresar propiedades Ω-regulares.

Sin embargo, estos y otros avances [6-7,19-23] resultan insuficientes para llevar a cabo por completo la transferencia al mundo industrial. Por ejemplo, FVS está limitado a lógicas temporales razonando únicamente sobre eventos en sistemas cerrados. En este sentido, la comunidad en Ingeniería de Software cree que una posible solución debe poder combinar varios condimentos: deben permitir la descripción parcial del comportamiento [15,26], la facilidad para especificar sistemas abiertos [26,35], se deben usar sintaxis simples y conocidas [23, 25], deben basarse en notaciones con suficiente poder expresivo [12,18,20,21], y deben soportar heterogeneidad (combinando declarativid para estar cerca de la especificación de los requerimientos [12] y notaciones operacionales basadas en autómatas o formalismossimilares[4]).

Dado este contexto, la línea de investigación detrás del presente proyecto es generar una sinergia entre tres líneas de investigación: descripción declarativa de propiedades, especificaciones parciales y síntesis de comportamiento para sistemas abiertos a partir deobjetivos.

Sistemas abiertos implica por un lado el diseño y manejo de algoritmos de control en Ingeniería de Software, y por otro, la posibilidad de contemplar el comportamiento esperado del ambiente, visión común dentro de la Ingeniería de Requerimientos [24]. Para mencionar un pequeño ejemplo de interacción con un ambiente, basta considerar el siguiente caso basado en el modelado de un sistema para el manejo de las luces interiores de un auto [16]. En el mismo, se busca evitar que la batería se descargue cuando funciona en un modo “caro”, que ocurre cuando las luces interiores se encienden y el auto está apagado. Para cumplir con este requerimiento la solución recae en pedir que el auto se encienda (y así la batería se recargue) entre dos funcionamientos consecutivos de la batería en

modo “caro”. Sin embargo, que el auto se encienda o no está fuera del control del sistema de luces, ya que forma parte de un comportamiento controlado por el ambiente o entorno. Este es sólo un pequeño ejemplo de la necesidad de extender FVS para especificar y sintetizar comportamiento en sistemas abiertos.

La síntesis de comportamiento ha sido atacada metodológicamente y algorítmicamente desde la síntesis y generación automática de controladores [5,9,14]. Se proveen mecanismos para diferenciar y distinguir el comportamiento que un componente puede hacer y controlar, y aquellas cosas que están fuera de su alcance: el comportamiento que proviene del entorno o ambiente que interactúa con el sistema. Sin embargo, existen limitaciones desde el punto de vista del lenguaje de especificación. Estas aproximaciones están fuertemente basadas en lógicas temporales (LTL) y algunas extensiones como fluents [3]. La expresividad de estos lenguajes ya ha sido desafiada por la comunidad, por lo que existe la necesidad de basarse en lenguajes de especificación con mayor poder expresivo. Asimismo, en estas técnicas no es posible determinar el controlador más general cuando se evalúan propiedades de tipo “liveness”, lo cual constituye una limitación conceptual de estas herramientas.

Por otro lado, existen otras aproximaciones basadas en especificaciones parciales que también han atacado el problema de expresividad en los lenguajes de especificación [6,15]. Estas aproximaciones se alejan de las estructuras lineales para enfocarse en estructuras “branching time”. Una característica de esta generación de especificaciones parciales es que posibilitan la descripción, manipulación y cómputo de posibles implementaciones que satisfacen los requerimientos planteados. En este contexto, es posible también el uso de MTS para tomar decisiones sobre la especificación operacional que más se ajuste a los requerimientos, particularmente en aquellas situaciones donde existe más de una implementación posible y se debe guiar al usuario para llegar a una única solución. Si bien el aporte de las estructuras “branching time” es importante a la hora de especificar comportamiento, existen limitaciones a la hora de trasladarlas a sistemas abiertos, donde se describen propiedades a ser satisfechas por un ambiente o entorno. En estas estructuras se describe el comportamiento de un componente, pero sin hacer distinción entre lo que el componente puede hacer y lo que puede controlar.

2. Líneas de Investigación y Desarrollo

En el presente trabajo se están explorando las siguientes líneas de investigación:

- Análisis de expresividad en notaciones operacionales y declarativas.
- Combinar distintas fuentes de especificación buscando obtener modelos operacionales y ejecutables.
- Aplicar el lenguaje desarrollado en casos de estudio de relevancia industrial como protocolos de comunicación y otros relacionados a los dominios de aplicación.
- Publicar los resultados parciales y finales en conferencias y revistas relativas al área de investigación
- Analizar especificaciones parciales basadas en estructuras "branching time".
- Diseñar extensiones a especificaciones parciales para manejar sistemas abiertos.
- Analizar técnicas de síntesis de controladores actuales.
- Potenciar lenguajes de especificación para síntesis de controladores.
- Generar extensión del lenguaje FVS en base a las extensiones detectadas para especificaciones parciales y síntesis de controladores.
- Desarrollar un nuevo lenguaje declarativo en base a las extensiones propuestas

- Generación y desarrollo de herramientas de software que den soporte a lo investigado.
- Modelado de comportamiento en arquitecturas desoftware.

3. Resultados Obtenidos/Esperados

En esta primera etapa los resultados alcanzados fueron:

- Publicaciones en congresos nacionales (JAIIO, CONAISSI).

-Combinar distintas fuentes de especificación: La traducción de escenarios y reglas FVS a autómatas de Buchi permitió combinar especificaciones declarativas realizadas en FVS con otras operacionales como aquellas basadas en notaciones de autómatas. Esto le provee a todo el entorno FVS una gran versatilidad ya que el comportamiento puede ser razonado y analizado desde requerimientos expresados en diversas fuentes y variantes. Esto es fundamental para continuar profundizando los casos de estudio, en particular haciendo énfasis en protocolos de comunicación.

- Potenciar crecimiento líneas de investigación y desarrollo dentro de la UNDAV: La difusión de los resultados logrados en el transcurso del año por parte de los docentes miembros del grupo permitió tanto el acercamiento de otros docentes como así de estudiantes de la carrera. En este sentido la investigación del proyecto permitió consolidar temas concretos para la realización de tesis por parte de los estudiantes, así como también la posibilidad de que realicen la PPS (Práctica Profesional Supervisada) dentro del proyecto de investigación. De la misma manera, también está contemplada una fuerte colaboración con el Laboratorio de Software Libre de la carrera de Ingeniería en Informática de la UNDAV, tanto para la obtención de casos de estudio como para el desarrollo de software.

- Aplicar el lenguaje desarrollado en casos de estudio de relevancia industrial como protocolos de comunicación y otros relacionados a los dominios de aplicación: Se trabajó en entornos de arquitectura de software, modelando interacciones complejas de comunicación entre componentes. La diversidad en el tipo de comunicaciones incluye comunicación entre artefactos de software, entre artefactos de software y hardware, así como también la interacción con capas intermedias de middleware como sensores.

- Desarrollar un nuevo lenguaje declarativo en base a las extensiones propuestas. Como primer paso se decidió estudiar la aplicabilidad de FVS para relevar y razonar sobre comportamiento arquitectónico. El avance actual de la investigación sobre este dominio permitió concluir que no fue necesario extender FVS de manera significativa en esta primera etapa, ya que fue lo suficientemente expresivo como para poder modelar comportamientoarquitectónicoderelevancia.

-Construcción de herramientas de software: En esta primera etapa se desarrollaron herramientas de software que dan soporte al lenguaje FVS. Se desarrolló la herramienta GTxFVS, que da la posibilidad al ingeniero de Software de trabajar en un entorno moderno para la especificación declarativa de comportamiento. De la misma manera, se implementó una herramienta basada en nociones orientadas a aspectos para la relevación dinámica de arquitecturas de software.

Los resultados esperados incluyen:

- Consolidación del lenguaje FVS en el dominio de arquitecturas de software.

- Inclusión de nociones de especificación parcial de comportamiento y lógicas de tipo “branching”.

- Síntesis de controladores bajo el contexto de sistemas abiertos.

- Publicaciones en revistas y congresos nacionaleseinternacionales.

- Dirección de tesis de licenciatura y supervisión de prácticas profesionales (PPS).

4. Formación de Recursos Humanos

Para este segundo año del proyecto están contempladas la incorporación de estudiantes avanzados para realizar tareas iniciales de investigación y en especial la realización de tesis de licenciatura. De la misma manera, se planea incorporar estudiantes para que realicen la PPS (Práctica Profesional Supervisada) dentro del proyecto de investigación. En particular, durante el primer cuatrimestre de 2017 un estudiante avanzado de la carrera, el estudiante Eric Loza, realizará su PPS desarrollando una herramienta de software dentro del marco del proyecto.

5. Bibliografía

[1] Clarke, E.M. and Grumberg, O. and Peled,D. Model Checking. MIT Press.1999

[2] M. Dwyer, G. Avrunin, and J. Corbett. Patterns in property specifications for finite-state verification. In Proceedings ICSE, volume 99, 1999.

[3] D. Giannakopoulou and J. Magee. Fluent model checking for event-based systems. In Proceedings of the 9th European software engineering conference, page 266. ACM, 2003.

[4] R. Smith, G. Avrunin, L. Clarke, and L. Osterweil. Propel: An approach supporting property elucidation. In ICSE, volume 24, pages 11-21, 2002.

[5] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive (1) designs. Lecture notes in computer science, vol. 3855, pp. 364-380, 2006.

[6] K. G. Larsen and B. Thomsen. A modal process logic. In LICS, pp 203–210. IEEE Computer Society, 1988.

[7] G. S. Walia and J. C. Carver. A systematic literature review to identify and classify software requirement errors. Inf. Softw. Technol., 51(7):1087-1109, 2009.

[8] Amalinda Post, Igor Menzel, Jochen Hoenicke, Andreas Podelski: Automotive behavioral requirements expressed in a specification pattern system: a case study at BOSCH. Requir. Eng. 17(1): 19-33 ,2012.

[9] B. Jobstmann, S. Galler, M. Weiglhofer, and R. Bloem, Anzu: a tool for property synthesis, in Proceedings of the 19th CAV'07, pp. 258-262, Springer-Verlag, 2007.

[10] Cohen, D.M. and Dalal, S.R. and Parelius, J. and Patton, G.C. The combinatorial design approach to automatic test generation. In IEEE Software Journal, volume 13, number 5, pages 83-88, 1996.

[11] Offutt, A.J. and Pan, J. Automatically detecting equivalent mutants and infeasible paths. In Software Testing, Verification and Reliability Journal, volume 7, number 3, pages 165-192, 1997.

[12] Fernando Asteasuain, Víctor Braberman. Specification Patterns can be formal and still easy, SEKE page 430-436 – 201.

[13] A. Alfonso, V. Braberman, N. Kicillof, and A. Olivero. Visual timed event scenarios. In 26th ICSE'04, pages 168-177, 2004.

[14] Nicolas D'Ippolito, Víctor Braberman, Nir Piterman, Sebastian Uchitel , Synthesising Non-Anomalous Event-Based Controllers for Liveness Goals. CM TRANSACTIONS ON SOFTWARE ENGINEERING AND METHODOLOGY, Volume in press - 2012

[15] German Sibay, Víctor Braberman, Sebastian Uchitel, Jeff Kramer , Distribution of Modal Transition Systems , 18th International Symposium on Formal Methods FM 2012, Volume in press – 2012.

[16] N. Noda and T. Kishi. An aspect-oriented modeling mechanism based on state diagrams. In 9th International Workshop on AOM, 2006.

[17] F. Asteasuain, V.Braberman. "Declaratively building behavior by means of scenario clauses". Requirements Engineering Journal. ISSN: 0947-3602. Diciembre 2015. DOI: 10.1007/s00766-015-0242-2

[18] Z. Wu. On the expressive power of qltl. In Proceedings of the 4th international conference on Theoretical aspects of computing, pages 467-481. Springer-Verlag, 2007.

[19] Kupferman, O. and Vardi, M. Module checking. In Computer Aided Verification, 75-86, 1996. Springler.

[20] Shoham Ben-David, Marsha Chechik, Arie Gurfinkel, Sebastián Uchitel: CSSL: a logic for specifying conditional scenarios. SIGSOFT FSE 2011: 37-47

[21] Ahmed Bouajjani, Yassine Lakhnech, Sergio Yovine: Model-Checking for Extended Timed Temporal Logics. FTRTFT 1996:306-326.

[22] I. Krka, Y. Brun, G. Edwards, and N. Medvidovic. Synthesizing partial component-level behavior models from system specifications. In ESEC/FSE '09, pages 305–314. ACM, 2009.

[23] Matthew B. Dwyer, George S. Avrunin, James C. Corbett: Patterns in Property Specifications for Finite-State Verification. ICSE 1999: 411-420

[24] A. V. Lamsweerde. Goal-oriented requirements engineering: A guided tour. In RE'01 –

[25] Margus Veanes, Colin Campbell, Wolfgang Grieskamp, Wolfram Schulte, Nikolai Tillmann, Lev Nachmanson: Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer. Formal Methods and Testing '08: 39-76

[26] David Harel: Statecharts: A Visual Formalism for Complex Systems. Sci. Comput. Program. 8(3): 231-274 (1987)

[27] DALAL, Siddhartha R., et al. Model-based testing in practice. En Proceedings of the 21st ICSE ACM, 1999. p. 285-294.

[28] F. Asteasuain, V.Braberman "Specification patterns: formal and easy". IJSEKE ISSN: 0218-1940. Vol. 25, No. 4 (2015) 669–700, DOI: 10.1142/S0218194015500060.