Un enfoque declarativo para modelar el comportamiento en sistemas reactivos

Fernando Asteasuain[1]

Departamento de Computación, FCEyN, UBA
fasteasuain@dc.uba.ar

1. Resumen

Existe consenso en la comunidad académica y de ingenieros de software que es fundamental entender, modelar y describir el comportamiento del software complejo desde etapas tempranas del desarrollo. El paradigma de descripción declarativa, basado en el modelado de las propiedades y objetivos esenciales de los objetos y agentes, posee características especialmente prometedoras para este tipo de desafíos [1]. Sin embargo, las alternativas existentes en este paradigma son lógicas temporales que poseen limitaciones prácticas y teóricas [18,9].

Asimismo, la verificación formal de propiedades sigue siendo en la actualidad uno de los mayores desafíos para la transferencia de tecnología de verificación de software como model checking. Los usuarios de estas técnicas deben enfrentar el desafío de expresar propiedades en el lenguaje formal usado en la herramienta de especificación. Dos de las aproximaciones más utilizadas son lógicas temporales como LTL, y notaciones operacionales basadas en autómatas. Ambas aproximaciones requieren usuarios "expertos" o con conocimientos avanzados en los mismos para poder expresar, describir y validar la propiedad de interés [12,9,21,18,5].

Todo esto indica la necesidad de contar con un lenguaje formal declarativo para expresar propiedades, que sea lo suficientemente esxpresivo y que permita realizar tareas de validación de manera simple e intuitiva. En esto contexto, el **objetivo global** de esta tesis es elaborar un enfoque de modelado declarativo, capaz de manejar distintos niveles de abstracción, con semántica precisa y clara, para modelar el comportamiento de sistemas reactivos. Se proyecta integrar este enfoque con otras áreas de verificación de software como la verificación, síntesis y model checking, la reconstrucción dinámica de arquitecturas, y el testing de conformidad. El **objetivo específico** de esta tesis es desarrollar un lenguaje de modelado declarativo, basado en notaciones gráficas (escenarios), capaz de modelar y describir el comportamiento de sistemas reactivos. El lenguaje contará con una semántica y sintaxis clara y precisa, con la posibilidad de realizar razonamiento automático, modelado incremental, y validación intuitiva de propiedades.

2. Problema

El modelado declarativo resulta un enfoque atractivo y natural para capturar requerimientos sobre comportamiento [15]. La declaratividad permite subespecificar comportamiento concentrándose únicamente en las propiedades relevantes, las cuales pueden agruparse según diferentes criterios como metas, agentes, temas, funcionalidades o conceptos. La opción clásica para notaciones declarativas en sistemas reactivos es lógica temporal. Sin embargo, en muchos casos, la descripción y validación de propiedades es una tarea compleja, aún para personas con conocimiento en el tema [18,9]. En este contexto la notación en escenarios representa una alternativa. Su representación gráfica facilita el entendimiento con usuarios y clientes, el comportamiento y casos de tests son más simples de confirmar o refutar, y complejos flujos de control e interacción pueden describirse fácilmente. Sin embargo, las notaciones de escenarios no han sido concebidas como lenguajes de especificación declarativos [6].

Respecto de la especificación de propiedades, un importante problema en la utilización de lenguajes formales como LTL, o notaciones basadas en autómatas es el grado avanzado de conocimiento que se requiere para poder expresar las propiedades de interés, y a su vez, validar que la propiedad está realmente describiendo la propiedad que el usuario tiene en mente. Este problema aún persiste cuando se utilizan patrones de especificación [9]. Si bien los patrones de especificación ofrecen una manera más amigable de expresar los requerimientos típicos, el usuario necesita validar la propiedad, esencialmente responder "¿Es esta la propiedad correcta?" Más aún, el usuario necesita poder comparar y analizar propiedades. Por ejemplo, al estudiar dos posibles propiedades candidatas preguntas típicas podrían ser ¿Cuál es más fuerte?, ¿Cuál impone más restricciones?, ¿Cómo se relacionan y porqué? Otra información valiosa para validar una propiedad es poder razonar sobre el comportamiento complementarios. Es decir, entender las distintas situaciones que llevan a la violación de una propiedad.

La evidencia indica que para realizar todas estas tareas de validación no alcanza con analizar la descripción en lenguaje natural del patrón elegido, sino que debe analizarse su traducción a un lenguaje formal [12,21,4]. Esto sugiere que el lenguaje de especificación debe ser fácil de usar, y lo suficientemente expresivo para permitir a usuarios expertos y no expertos usarlo apropiadamente [17,12]. Lenguajes formales como LTL o notaciones basadas en autómatas no logran satisfacer por completo la validación de propiedades [4,12] . La manipulación de complejas formulas LTL o autómatas no es una tarea trivial. Aún para casos sencillos se requieren la asistencia de herramientas expuestas a operaciones costosas y sofisticadas.

Este contexto ilustra la importancia y necesidad de contar con un lenguaje declarativo formal para expresar propiedades. Dicho lenguaje formal debe manejar correctamente tareas de validación. Es decir, las especificaciones resultantes deben ser concisas, fáciles de comparar y modificar. Asimismo, debe ser sencillo poder razonar sobre el complemento de cada propiedad.

3. Trabajo relacionado

TimeEdit [20] y GIL (Graphical Interval Logic) [8] son dos lenguajes gráficos de especificación basados en diagramas con lineas de tiempo que no poseen un orden parcial de eventos. TimeEdit está particularmente enfocado en capturar cadenas complejas de eventos [5], y cuenta con una noción restringida de escenarios basada en reglas (eventos que deben ocurrir si todos los eventos previos ocurrieron). Estas restricciones hacen que especificar propiedades sobre eventos pasados, o eventos que sucedieron dentro de un cierto alcance sean más dificiles de especificar y entender. GIL provee también la posibilidad de definir intervalos en la ocurrencia de eventos. Sin embargo, aquellas propiedades que involucren varios eventos compuestos requieren un anidamiento importante de operadores, debilitando el proceos de validación de las propiedades. Propel [21] es una herramienta utilizada para la especificación de propiedades. La misma está equipada con dos notaciones: una notación basada en lenguaje natural con restricciones, y una notación operacional basada en autómatas finitos. Propel cuenta con varias limitaciones a la hora de poder especificar patrones de especificación [9]. Patrones complejos que involucren una secuencias de eventos no son posibles de especificar en dicho enfoque. Tampoco es posible especificar en Propel la repetición de comportamiento en patrones. Finalmente, Propel impone restricciones al tipo de evento que delimita el alcance de una propiedad. Otros formalimos visuales basados en notaciones operacionales como Message Sequence Charts (ver [19,22,11]) también han sido propuestos como especificaciones basadas en escenarios. El lenguaje desarrollado en esta tesis comparte con estas aproximaciones la idea de utilizar órden parcial de eventos para describir escenarios. Sin embargo, la investigación de esta tesis difiere en varios aspectos. En primer término, esta investigación está enfocada en expresar propiedades para ser verificadas contra un modelo o sistema bajo análisis, y no en crear un lenguaje de modelado ejecutable para diferentes étapas del proceso de desarrollo. Otra limitación de estos enfoques es que no son notaciones sumamente flexibles, ya que es dificil razonar sobre comportamiento pasado, o comportamiento que ocurre entre medio de dos eventos. Finalmente, es importante destacar que ninguna de las aproximaciones mencionadas cuenta con notaciones suficientes poderosas para poder efectuar razonamientos deductivo para la comparación de propiedades ni para razonar sobre comportamiento complementarios.

4. Marco de trabajo

Como primera instancia se definieron atributos de calidad deseables en un lenguaje formal para una correcta especificación y validación de propiedades. Dichos atributos son: *Sucinto*, *Facilidad de Verificación*, y *Modificabilidad*. Sucinto se refiere a qué tan conciso puede ser expresar una propiedad. Este atributo es esencial para poder hacer más sencilla la validación de propiedaes. La facilidad de verificación se subdivide a su vez en dos subatributos: facilidad de comparación, y de complemento. El primero establece que las propiedades deben ser fáciles

de comparar, de distinguir, y de entender la relación entre ellas. La segunda se refiere a que debe ser sencillo entender las distintas situaciones que llevan a la violación de la propiedad. Esta información es de gran utilidad a la hora de validar una propiedad. Finalmente, modificabilidad se refiere a la habilidad para poder manipular una propiedad para poder adaptarla a nuevas contextos de aplicación.

Luego, en [4] se presentan los lineamientos generales del lenguaje declarativo diseñado para cubrir todos los objetivos mencionados, denominado FVS (FeatherWeight Visual Scenarios). FVS, a pesar de su simpleza, es lo suficientemente poderoso como para describir todos los patrones de especificación. Por un lado, su naturaleza gráfica y visual ayuda al usuario en concentrarse únicamente en las propiedades y no tratar con las complicaciones de su formalización. Los escenarios son construidos usando una cantidad minimal de operadores simples, obteniendo así especificaciones concisas y compactas. Relaciones lógicas y semánticas pueden fácilmente deducirse directamente de los escenarios, aumentando la posibilidad de razonar sobre las propiedades. El lenguaje cuenta con la posibilidad de construir automáticamente anti-escenarios, los cuales ayudan al usuario en la especificación de propiedades examinando el comportamiento que lleva a una violación de una propiedad. Por último, la especificación de propiedades es flexible, y puede adaptarse fácilmente a diferentes contextos de aplicación.

Para validar la aplicabilidad y usabilidad de FVS respecto de los atributos definidos, se tomó como caso de estudio los patrones de especificación propuestos en [10]. Los patrones de especificación se describen detallando por un lado el comportamiento capturado por el patrón, como también el alcance o la porción de la ejecución del sistema donde la propiedad debe valer. Este caso de estudio es altamente representativo. Un estudio presentado en [9] analiza al menos 555 especificaciones de al menos 35 fuentes diferentes, demostrando que el 92 % de las propiedades podía ser descripta a través de los patrones. En [4] se comparó las especificaciones de todos los patrones de especificación de FVS contra las formulas lógicas LTL propuestas en [9] y contra notaciones basadas en autómatas. Como conclusión de dicha comparación se puede establecer que las espeficaciones en FVS son más concisas, mas simples de comparar, analizar y modificar. Esencialmente, FVS logra manejar apropiadamente todas las tareas que involucra la validación de propiedades.

Para incorporar nociones avanzadas de modularización en [2,3] se explora la posibilidad de definir a FVS como un lenguaje de modelado orientado a aspectos, buscando introducir modelos de *joinpoints* más flexibles. En los últimos años, la orientación a aspectos ha surgido como un enfoque interesante para tratar con la complejidad en la descripción de entidades de software. Sin embargo, algunos autores han señalado dificultades para aplicar la filosofía orientada a aspectos en notaciones operacionales [14,13]. Muchas aproximaciones orientadas a aspectos terminan recayendo en mecanismos de composición (weaving) sintácticos, sin una semántica clara [14]. FVS ataca estos problemas brindando una mayor flexiblidad para desacoplar la interacción entre los aspectos y el sistema bajo análisis.

Finalmente, parte de la metodología a cubrir incluye caracterizar el tipo de propiedades expresables en FVS. En particular, investigar la siguiente inquietud: ¿FVS $\subseteq$ LTL?. Para tal punto, es preciso investigar la caracterización propuesta por [16], donde agrupa a las propiedades LTL en seis categorías: *Safety*, *Guarantee*, *Obligation*, *Response*, *Persistence*, y *Reactivity*. En este sentido, se buscará aumentar la expresividad de FVS introduciendo eventos auxiliares, los cuales permitirán introducir niveles de abstracción, con la posibilidad de definir lenguajes ω-regulares, una característica sobresaliente al comparar FVS contra lógicas temporales. Una caracterización inicial de esta extensión puede verse en [1].

5. Desafíos

Uno de los principales desafíos de la presente tesis surge de buscar cumplir con dos objetivos en principio en conflicto, como son la usabilidad y la rigurosidad formal. Es decir, diseñar un lenguaje formal para especificar propiedades, pero donde a su vez, sea sencillo e intuitivo realizar tareas de validación. En este sentido, uno de los desafíos salientes es determinar el nivel de expresividad necesario en dicho lenguaje. Esto implica la comparación de expresividad contra otros mecanismos formales como lógicas temporales o notaciones basadas en autómatas, requiriendo demostraciones formales de inclusión de lenguajes. A su vez, la posibilidad de contar con algorítmos de síntesis a partir de especificaciones constituye también un punto saliente. El desarrollo de un algoritmo de traducción de especificaciones FVS a autómatas de Buchi constituye un punto importante en la presente investigación. Existen diferentes dificultades técnicas al respecto, ya que la manipulación de autómatas de Buchi no es trivial, involucrando operaciones expuestas a explosión de estados, y con algoritmos con órdenes prohibitivos de complejidad, como por ejemplo, la complementación.

6. Próximos Pasos

Como se estableció anteriormente, el próximo paso inmediato es incorporar a la notación de FVS la posibilidad de definir eventos auxiliares. Los mismos permitirán introducir distintos niveles de abstracción en las especificaciones, logrando así la posibilidad de denotar cualquier lenguaje ω-regular. Dicho poder expresivo es una característica sobresaliente respecto de la mayoría de lenguajes formales utilizados para la especificación de propiedades. Este sería el salto de expresividad necesario para pasar de describir propiedades a especificar comportamiento. Otro paso futuro importante para la consolidación de la tesis consiste en un algoritmo de traducción de los escenarios FVS a autómatas de Buchi. Dicha traducción permitiría un interesante algoritomo de síntesis, introduciendo así la posibilidad de realizar mecanismos avanzados de análisis automático. Finalmente, se desea explorar la interacción con sistemas abiertos, especialmente para sintetizar Interface Autómata [7], teniendo en cuenta la controlabilidad de los eventos.

Referencias

1. F. Asteasuain and V. Braberman. Behavioral modeling. Technical Report 03-09, Computer Science Department – School of Science - UBA, 2009.
2. F. Asteasuain and V. Braberman. On the need for a declarative and incremental behavioral modeling. In *LA-WASP*, 2009.
3. F. Asteasuain and V. Braberman. Exploring Visual Scenarios as an aspect-oriented modeling language. In *ASSE*, 2010.
4. F. Asteasuain and V. Braberman. Specification patterns can be formal and also easy. In *Software Engineering and Knowledge Engineering (SEKE)*, 2010.
5. M. Autili, P. Inverardi, and P. Pelliccione. Graphical scenarios for specifying temporal properties: an automated approach. *ASE*, 14(3):293–340, 2007.
6. V. Braberman, N. Kicillof, and A. Olivero. A scenario-matching approach to the description and model checking of real-time properties. *IEEE TSE*, 31(12):1028–1041, 2005.
7. L. De Alfaro and T. Henzinger. Interface automata. *ACM SIGSOFT Software Engineering Notes*, 26(5):120, 2001.
8. L. Dillon, G. Kutty, L. Moser, P. Melliar-Smith, and Y. Ramakrishna. A graphical interval logic for specifying concurrent systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 3(2):131–165, 1994.
9. M. Dwyer, G. Avrunin, and J. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st International Conference on Software Engineering ICSE*, volume 99, 1999.
10. M. Dwyer, M. Avrunin, and M. Corbett. Patterns in property specifications for finite-state verification. In *21th ICSE*, pages 411–420, 1999.
11. D. Harel and R. Marelly. Playing with time: On the specification and execution of time-enriched lscs. In *MASCOTS '02*, pages 193–202. IEEE Computer Society.
12. G. Holzmann. The logic of bugs. *ACM SIGSOFT Software Engineering Notes*, 27(6):87, 2002.
13. S. Katz. Diagnosis of harmful aspects using regression verification. In *FOAL: Foundations Of Aspect-Oriented Languages*, pages 1–6, 2004.
14. S. Katz. Aspect categories and classes of temporal properties. In *Trans. Aspect-Oriented Softw. Develop*, pages 106–134, 2006.
15. A. V. Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *RE'01 - International Joint Conference on RE*, 2001.
16. Z. Manna and A. Pnueli. A Hierarchy of Temporal Properties. In *Proceedings of the... Annual ACM Symposium on Principles of Distributed Computing*, page 205. Association for Computing Machinery, 1987.
17. D. Paun and M. Chechik. Events in linear-time properties. In *Proceedings of 4th International Conference on Requirements Engineering*. Citeseer, 1999.
18. R. W. R. and K.Viggers. Implementing protocols via declarative event patterns. In *ACM Sigsoft International Symposium on FSE(FSE-12)*, pages 158–169, 2004.
19. B. Sengupta and R. Cleaveland. Triggered message sequence charts. In *SIGSOFT FSE*, pages 167–176, 2002.
20. M. Smith, G. Holzmann, and K. Etessami. Events and constraints: A graphical editor for capturing logic requirements of programs. In *Proceedings of the 5th RE*, pages 14–22. Citeseer, 2001.
21. R. Smith, G. Avrunin, L. Clarke, and L. Osterweil. Propel: An approach supporting property elucidation. In *ICSE*, volume 24, pages 11–21, 2002.
22. S. Uchitel, J. Kramer, and J. Magee. Negative scenarios for implied scenario elicitation. In *Proc. of FSE '02*, pages 109–118. ACM Press, 2002.