Análisis para el Desarrollo Multiparadigma

Silvia Amaro, Pablo Quiroga
{samaro, pquiroga}@uncoma.edu.ar
Claudio Vaucheret, Lidia López, Daniel Dolz
{vaucheret, lidiamlopez, daniel.dolz}@gmail.com
Andrea Granados, Ingrid Godoy, Viviana Sánchez, Maximiliano Klemen
{agranados80, ingriduncoma, sanchez.viviana, maximilianoklemen}@gmail.com
Paola Pérez
ppercat@hotmail.com

Dpto. de Ciencias de la Computación
Facultad de Economía y Administración
Universidad Nacional del Comahue, Argentina

1. Contexto

La línea de investigación orientada al desarrollo multiparadigma se enmarca dentro del proyecto de investigación ”Técnicas Avanzadas y Análisis para el Desarrollo Multiparadigma”

2. Resumen

En el proceso de desarrollo de software, es especialmente importante lograr los resultados espe-rados en el rango de tiempo esperado y con la calidad esperada, aplicando las técnicas y prácticas mas apropiadas al problema planteado. Para ello es necesario ser capaces de identificar la naturaleza del problema a resolver para elegir las herramientas y técnicas que mejor se adapten a la situación, teniendo en cuenta que diferentes tipos de problemas pueden ser atacados utilizando los paradigmas apropiados. Debido a que usualmente los problemas complejos tienen una naturaleza multidimensional, en la mayoría de los casos es apropiado utilizar una aproximación multiparadigma. Nuestro objetivo es el estudio de técnicas y análisis globales de programas dentro del contexto del desarrollo multiparadigma, tendiendo a establecer modelos, procesos y técnicas para mejorar el desarrollo de sistemas en un ambiente multiparadigma.

Palabras clave: Programación multiparadigma, Orientación a Componentes, Orientación a Aspectos, Orientación a Servicios, Análisis estático, Análisis dinámico.

3. Introducción

En el proceso de desarrollo de software las abstracciones juegan un rol central. Una abstracción se enfoca en la esencia del problema y excluye los detalles especiales. El desarrollo de software es fundamentalmente una actividad humana de forma que, a pesar de que en la actualidad es amplia-mente soportado por herramientas automáticas, aún es influenciado por experiencias personales, tradiciones, convenciones y costumbres. Estos patrones de comportamiento tienen un impacto importante sobre el proceso de desarrollo de software determinando la forma en que los desarrolladores

seleccionan y forman las abstracciones.

En tecnologías de software un paradigma representa las directivas para crear abstracciones. El paradigma es el principio por el cual un problema puede ser comprendido y descompuesto en componentes manejables. Un paradigma de programación es el modo de conceptualizar lo que significa realizar una computación y cómo las tareas a llevarse a cabo sobre una computadora deben ser estructuradas y organizadas. El lenguaje en el cual un programador piensa un problema influirá en modo básico fundamental la forma en la cual un algoritmo es desarrollado.

Algunos paradigmas ya establecidos son:

- Programación Imperativa o Procedural: Modelo tradicional de computación, está estrechamente relacionado con el modelo de ejecución en el hardware subyacente.
- Programación Orientada a Objetos: Representación de los problemas intuitivamente. La computación se realiza a través de objetos comunicándose entre ellos y requiriendo que otros objetos realicen acciones.
- Programación Funcional: Tratamiento de las variables y sus valores tratados como entidades únicas. Una vez creados, los valores no son modificados, sino que son transformados en nuevos valores independientes de los originales.
- Programación Lógica: Basado en el cálculo de predicados. La programación es declarativa, el programador simplemente provee una serie de aserciones de hechos, un conjunto de reglas de inferencia y consultas.
- Programación orientada al Acceso: Uso de efectos laterales o ”demonios”que son vinculados a las variables de manera que ciertas funciones sean realizadas en el momento en que las variables son manipuladas.
- Programación de restricciones: Especificación de una secuencia de restricciones que deben ser mantenidas durante la ejecución del programa.
- Programación literal: Combinación de documentación y código fuente en un único archivo con el fin de mantener la correspondencia entre la documentación y el código mismo.

Además, se puede identificar: (a) programación visual, (b) programación paralela, (c) programación orientada a aspectos, (d) programación orientada a componentes, las que se basan en alguno o varios de los paradigmas anteriores, (e) programación orientada a servicios.

Ningún paradigma es capaz de resolver todos los problemas de forma sencilla y eficiente, por lo tanto es útil poder elegir entre distintas técnicas de programación dependiendo del tipo de problema. En este escenario el desarrollo multiparadigma tiene como principal objetivo proveer al programador con un conjunto variado de herramientas de manera que cuando enfrente un proble-ma no se vea forzado a utilizar una única técnica de programación, sino que tenga la libertad de seleccionar la técnica de solución que mejor se adapte a las características del problema a resolver.

En particular, un lenguaje que se está tomando como caso de estudio y plataforma de desarrollo dada sus características únicas es *Ciao Prolog* [9, 10]. *Ciao* es un lenguaje multiparadigma basado en expansiones sintácticas que implementan distintos paradigmas mediante la técnica de compilación de control.

La aparición de la programación orientada a objetos permitió comenzar a desarrollar software con una visión diferente. En este escenario surge la Programación Orientada a Aspectos (AOP), que propone una mejor modularización

[11]. AOP trabaja con paradigmas y lenguajes ya existentes para mejorar su expresividad y funcionalidad, y por ser un desarrollo que continúa el paradigma de Programación Orientada a Objetos soporta la descomposición orientada a aspectos, orientada a objetos y funcional [12, 13]. Es una metodología de programación que crece y evoluciona rápidamente.

Las técnicas de orientación a aspectos, en conjunto con técnicas de metadatos pueden ser utilizadas para inspeccionar un sistema y extraer los datos necesarios para diseñar heurísticas que sean utilizadas por los mecanismos de reflexión y orientación a aspectos para manejar la evolución [6].

La programación orientada a componentes se propone producir componentes de software para un mercado y una futura composición. Los compositores son terceras partes, posiblemente usuarios finales, que no están habilitados a realizar cambios. Esto requiere que componentes creadas de forma independiente puedan interoperar y además requiere de especificaciones que ubiquen al compositor en posición de decidir qué puede ser compuesto y bajo qué condiciones. En este sentido la coordinación y adaptación son puntos clave, y se enfocan en las interacciones entre entidades computacionales y los problemas que surgen cuando las entidades no concuerdan, respectivamente [4]. Los protocolos de interacción pueden ser complementados con el uso de aspectos para soportar la evolución de los componentes [1, 2]. En el mismo sentido existen varias propuestas para utilizar las tecnologías de la orientación a aspectos en el contexto de los servicios web [5].

4. Líneas de Investigación y Desarrollo

Para lograr los objetivos planteados se están abordando distintas temáticas en el contexto del desarrollo multiparadigma, como:

- Análisis global estático de programas no dependiente del paradigma. Uno de los tipos de análisis realizado en Ciao Prolog que puede ser transparente al paradigma utilizado, es la inferencia de tipos por medio de análisis estático ”top-down”de programas basados en interpretación abstracta que tiene gran aplicación en la optimización y depuración de programas [3, 8]. La interpretación abstracta de programas es una tecnología que permite el análisis estático de programas para obtener propiedades de la ejecución de los mismos sin ejecutar realmente los programas.

- Análisis y adecuación de técnicas orientadas a aspectos. En general, al trabajar con lenguajes de AOP, la lógica de los aspectos se entrelaza en la aplicación destino, y en esta situación múltiples aspectos, desarrollados de forma independiente pueden producir comportamiento inesperado en el sistema. Este problema ha sido identificado como el problema de la interacción de características (Feature Interaction Problem). La interacción entre distintos aspectos puede darse cuando especifican comportamiento en el mismo punto, pero también cuando los aspectos no comparten el punto de encuentro. Además, los aspectos a menudo ejercen amplias influencias sobre los intereses, por ejemplo modificando su semántica, estructura o comportamiento. Estas dependencias entre elementos aspectuales y no aspectuales pueden derivar en interacciones tanto deseables como no deseables[7].

- Análisis de propiedades estáticas y dinámicas en orientación a componentes. Es interesante determinar si la evolución basada en aspectos preserva las nociones fundamentales de co-rrectitud, en particular la compatibilidad y sustitutabilidad de componentes. ¿Se preserva la naturaleza de caja negra de los componentes?, ¿cómo afecta a la naturaleza de caja negra de un componente y a su composición el uso de un paradigma de programación declarativo?.

- Extensiones de lenguajes para incorporar paradigmas no nativos. Se está realizando un análisis de los lenguajes que tienen características multiparadigma, para definir un entorno comparativo de los mismos. Interesa estudiar la posibilidad de extender algunos lenguajes, incorporando características multiparadigma.

5. Resultados esperados

El desarrollo multiparadigma permite la construcción de sistemas mediante la selección del paradigma apropiado en cada situación. El producto de la presente propuesta puede ser de utilidad para el desarrollo de aplicaciones en dominios particulares y que deban cumplir con propiedades particulares, dando la posibilidad de evaluar y seleccionar las mejores técnicas y paradigmas para la construcción de sistemas más eficientes, seguros y de mejor calidad.

6. Formación de Recursos Humanos

El mayor impacto del presente proyecto se centra en la formación de recursos humanos, consolidación de grupos de investigación e interacción entre grupos interdisciplinarios. Permitirá a parte de los autores dar sus primeros pasos en investigación. Se prevee continuar con trabajos conjuntos con docentes de la Universidad de Malaga y la Universidad de Huelva, España. Se aspira a comenzar un trabajo conjunto con docentes de la Universidad Libre de Bruselas (Vrije Universiteit Brussel), y con docentes de la Universidad de Madrid. En particular las líneas de investigación propuestas posibilitarán a sus integrantes la conclusión de estudios de grado, conclusión de estudios de posgrado en curso y la iniciación de nuevos estudios de posgrado.

Referencias

[1] Guido Soldner and Rudiger Kapetza. *AOCI: An Aspect-Oriented Component Infraestructure*. Workshop on Component Oriented Programming WCOP, ECOOP 2007, Berlin, Alemania.

[2] Angel Nunez, Jacques Nayé. *A Seamless Extensión of Components with Aspects using Protocols.* Workshop on Component Oriented Programming WCOP, ECOOP 2007, Berlin, Alemania.

[3] Debra J. Richardson. *Static Análisis: Analysis of Models, Data flow analices, Finite-state verification.* http://www.ics.uci.edu/08-StaticAnalysis.pdf, University Of California. 2000

[4] S. Amaro, E. Pimentel and A.M. Roldán, *Reo Based Interaction Model*, International Workshop on Formal Aspects of Component Software, FACS´05, UNU-IIST, Macao, China, Octubre 2005.

[5] Florian Irmert, Marcus Meyerhofer, Markus Weiten. *Towards Runtime Adaptation in SOA Environment.* Workshop on Reflection, AOP and Meta-Data for Software Evolution, RAM-SE, ECOOP 2007, Berlin, Alemania.

[6] Dong Ha Nguyen and Mario Südholt. *Property-Preserving Evolution of Components Using VPA-Based Aspects. Workshop on Reflection, AOP and Meta-Data for Software Evolution*, RAM-SE, ECOOP 2007, Berlin, Alemania.

[7] Pablo Quiroga. Control-Flow Interaction in Aspect-Oriented Programming. Tesis Master of Science in Computer Science, Nantes, Francia. Septiembre 2007.

[8] R. Barbuti, R. Giacobazzi, and G. Levi. *A general framework for semantics-based bottom-up abstract interpretation of logic programs.* ACM Transactions on Programming Languages and Systems, 15(1):133-181, 1993.

[9] P. Van Hentenryck, A. Cortesi, and B. Le Charlier. *Type analysis of prolog using type graphs.* Journal of Logic Programming, 22(3):179-209, 1995.

[10] C. Vaucheret and F. Bueno. *More precise yet efficient type inference for logic programs.* In International Static Analysis Symposium, number 2477 in LNCS, pages 102-116. Springer-Verlag, September 2002.

[11] Robert E. Filman, Tzilla Elrad, Siobhán Clarke, and Mehmet Ak¸sit, editors. *Aspect-Oriented Software Development.* Addison-Wesley, Boston, 2005.

[12] K. Mehner and A. Wagner. *An assessment of aspect language design.* Young Researchers Workshop, First International Symposium on Generative and Component-Based Software Engineering, September 1999.

[13] Éric Tanter and Jacques Noyé. *A Versatile Kernel for Multi-Language AOP.* In Proceedings of the ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering (GPCE 2005), LNCS, Springer-Verlag, Tallin, Estonia, September, 2005.