Diseño y Construcción de Lenguajes Específicos del Dominio

Mariano Luzza[(1)], Mario Berón[(1)], Germán Montejano[(1)], Pedro Rangel Henriques[(2)], Maria J. Pereira[(3)]
[(1)]Departamento de Informática/Facultad de Ciencias Físico Matemáticas y Naturales/Universidad Nacional de San Luis
Ejército de los Andes 950 – San Luis – Argentina
email: {mberon,mluzza,gmonte}@unsl.edu.ar
[(2)]Pedro Rangel Henriques
Departamento de Informática/Universidade do Minho
Braga-Portugal
email: pedrorangelhenriques@gmail.com
[(3)]Deparamento de Informática/Instituto Politécnico de Bragança
Bragança-Portugal
mjoao@ipb.pt

RESUMEN

En la actualidad existen numerosos problemas, que si bien pueden ser solucionados con herramientas de propósito general, es más apropiado abordarlos con aplicaciones específicas para ese dominio. En este contexto se encuentran los Lenguajes Específicos del Dominio (DSL-Domain Specific Language).
Un DSL es un conjunto reducido de construcciones y operaciones que brindan una mayor expresividad y optimización para un dominio particular.
La construcción de DSLs no es una tarea trivial ya que implica tres fases muy importantes como lo son: Análisis, Diseño e Implementación. Si bien estas fases coinciden con un proceso de desarrollo de software tradicional tienen otras particularidades relacionadas con los DSL que introducen nuevos desafíos para la investigación y desarrollo de los mismos.
En este artículo se presenta una línea de investigación que tiene por objetivo dilucidar las actividades involucradas en el proceso de construcción de DSL.

Palabras Claves: Lenguaje Específico del Dominio, Compiladores, Generador de Compiladores.

CONTEXTO

La línea de investigación descripta en este artículo se encuentra enmarcada en dos proyectos de investigación. El primero es *Ingeniería de Software: Conceptos Métodos Técnicas y Herramientas en un Contexto de Ingeniería de Software en Evolución* de la Universidad Nacional de San Luis.
El segundo es un proyecto bilateral entre la Universidad Nacional de San Luis y la Universidade do Minho-Portugal. Dicho proyecto se denomina: *Quixote: Desarrollo de Modelos del Dominio del Problema para Inter-relacionar las Vistas Comportamental y Operacional de Sistemas de Software.* Quixote fue aprobado por el Ministerio de Ciencia, Tecnología e Innovación Productiva de la Nación (MinCyT) y la Fundação para a Ciência e Tecnología (FCT) de Portugal. Ambos entes soportan económicamente la realización de diferentes misiones de investigación desde Argentina a Portugal y viceversa.

1. INTRODUCCIÓN

La ingeniería de software se encarga de resolver problemas a través de métodos, técnicas y herramientas. Dichos métodos, técnicas y herramientas están especificados/programados en diferentes tipos de lenguajes de propósito general (GPL), ampliamente usados por su versatilidad para resolver una gran diversidad de problemas. Sin embargo, existen numerosas aplicaciones en donde el uso de lenguajes de propósito general no es el más adecuado. Esta afirmación se basa en que las soluciones propuestas producen un código poco legible, ya que seguramente no está expresado en términos del dominio de la solución, y se necesitan realizar muchas tareas que obscurecen la solución del problema, como llamadas a procedimientos o cálculos que no están en forma primitiva en el lenguaje. Los

GPL se caracterizan por: i) Tener un mayor soporte con manuales y herramientas; ii) Ser ampliamente usados por los programadores; iii) Ser generalmente de más bajo nivel que los DSL [4], por consiguiente poseen más detalles sintácticos y están más sujetos a errores.

A veces parece que los programadores encriptan el conocimiento del dominio y los requisitos de sus clientes, ya que literalmente, ocultan información valiosa en una montaña de código. El problema está en que, una vez que el código está escrito, cuando los desarrolladores tienen que hacer nuevos módulos o cambios, deben modificar el código en sí mismo, lo cual puede introducir errores y es además una tarea tediosa y lenta.

Ciertos dominios resultan muy difíciles de llevar a cabo con lenguajes de propósito general. Como por ejemplo las gramáticas, puede resultar muy sencillo realizar una especificación con BNF y claramente no es natural abordar dicho problema con un GPL. Los DSL proveen una solución adecuada a la problemática descripta previamente.

Los DSL se caracterizan por: i) Ser lenguajes más declarativos, más descriptivos, de más alto nivel, más comprensibles y más fáciles de aprender, ii) Normalmente son especificados con gramáticas pequeñas con pocos detalles de sintaxis y por consiguiente las especificaciones/programas escritos en un DSL están menos propensos a errores; iii) Los usuarios de un DSL crean más fácilmente la correspondencia entre el programa y el problema que se pretende resolver; iv) Permiten una mayor productividad en el contexto del dominio para el cual fueron definidos y v) Las especificaciones o programas son más fáciles de mantener.

A pesar de ser lenguajes generalmente pequeños su creación y desarrollo no es una tarea trivial y merece ser abordada como una línea de investigación[2,6,7].

Este artículo está organizado como sigue. La sección 2 tiene tres finalidades importantes. La primera explica qué es un DSL, la segunda menciona los usos comunes de los DSL y la tercera muestra ejemplos de DSL[3]. La sección 3 expone los resultados de la investigación. La sección 4 describe la formación de recursos humanos llevada a cabo en el contexto de la línea de investigación presentada en este artículo.

2. LENGUAJES ESPECÍFICOS DEL DOMINIO

2.1. Definición

Un Lenguaje Específico del Dominio (DSL) es un conjunto reducido de construcciones y operaciones que brindan una mayor expresividad y optimización para un dominio particular. Según Hudak [1] un DSL es “la última abstracción”, que captura precisamente la semántica de un dominio de aplicación. Algunos DSL bien conocidos incluyen SQL y expresiones regulares. Claramente cada uno es mejor que un lenguaje de propósito general para representar operaciones sobre, base de datos y cadenas respectivamente, pero no lo harán muy bien para describir soluciones fuera de su dominio. Algunas industrias poseen también sus propios DSL. Por ejemplo, en telecomunicaciones, los lenguajes de descripción de llamadas son ampliamente utilizados para especificar la secuencia de estados en una llamada telefónica, y en la industria de viajes, otro lenguaje podría ser usado para describir reservas de vuelo. Otras áreas donde también se podrían usar DSL incluyen el plan de rutas de navegación de un sitio web, diagramas de conexión de componentes electrónicos o un árbol genealógico, entre otras.

2.2. Utilización

Los usuarios de los DSL crean modelos que luego se compilan o se traducen a otros artefactos. Es común hallar DSL cuyo uso es generar código de programa en otro lenguaje o un ejecutable, pero también son utilizados para generar otros artefactos como un esquema de visualización para ciertos datos. Cuando se define un DSL, se pueden definir plantillas que lean un modelo del DSL y generen ejecutables, fuentes de otros lenguajes, archivos de texto u otros artefactos. Por ejemplo, se podría tener una plantilla que tome varías relaciones de parentesco simples como padre, hijo y hermano y genere otras relaciones deducibles como abuelo, sobrino y primo. Generalmente, los DSL son creados cuando un grupo de usuarios (no

necesariamente desarrolladores) tienen que generar código similar para varios productos. Por ejemplo, una compañía que se especializa en sistemas de manejo de valijas podría diseñar un DSL para el seguimiento de valijas con el cual podrían generar parte del código de cada producto. El principal beneficio de los DSL, es que pueden ser más sencillos de entender por los usuarios, tanto del DSL como quizás también de los que usarán el producto generado por el DSL, ya que se manejan conceptos en términos del dominio. Además el código (u otro artefacto generado) es más confiable, gracias a la automatización y refactorización de algunas tareas. En un DSL, la representación es más sencilla, sólo se describe como resolver el problema en términos del dominio y no hay construcciones especiales de código para poder completar el algoritmo como ocurriría en un GPL. Gracias a esto, es más sencillo también introducir cambios en el código si hay cambios en la especificación del problema. A su vez, quizás no sea necesario un programador para utilizar el DSL, ya que estará en términos del dominio y que cualquier usuario del entorno debería manejar. Los programadores podrían seguir siendo necesarios para crear la aplicación, sin embargo, la aplicación está escrita de una manera que permite a los expertos en la materia asistir en la lógica de negocio. La habilitación del experto en el dominio aumenta en gran medida la eficiencia del mantenimiento de una aplicación a medida que hay cambios en las especificaciones.

2.3. Ejemplos

En los siguientes párrafos se ilustra la necesidad de este tipo de lenguajes.

2.3.1. BNF

Un DSL conocido por programadores es el BNF. Este lenguaje simplifica la representación de gramáticas libres del contexto. Generalmente es usada para describir las sintaxis de los lenguajes de programación que se usan cotidianamente, siendo así una meta-sintaxis. Claramente este es un DSL donde su dominio son las sintaxis y se puede ver que es más sencillo definir un lenguaje de programación especificando su sintaxis en BNF que directamente construyendo el analizador sintáctico y el compilador. Además, un documento BNF se puede utilizar como entrada para un generador de analizadores sintácticos, automatizando, estandarizando y optimizando el proceso.

2.3.2. BPMN

Business Process Modeling Notation es una notación gráfica para modelar procesos de negocio, en un formato de flujo de trabajo (workflow). El principal objetivo de BPMN es proveer una notación estándar que sea de fácil lectura y entendible por parte de todos los involucrados e interesados del negocio. En síntesis BPMN tiene la finalidad de servir como lenguaje común para cerrar la brecha de comunicación que frecuentemente se presenta entre el diseño de los procesos de negocio y su implementación. Así se ve que para BPMN, su dominio específico son los procesos de negocio y, como muchos otros, que su objetivo es acercar el lenguaje al dominio haciendo que todos puedan entenderlo en términos del dominio.

2.3.3. VHDL

El lenguaje de descripción de hardware VHDL, es otro DSL cuyo propósito es simplificar la tarea de describir el diseño de dichos circuitos. Claramente es mucho más sencillo para los electrónicos trabajar con este lenguaje en términos de lo que dominan, que hacerlo en Pascal o Ada, lenguajes de propósito general en los que se basó VHDL. En VHDL se maneja de mejor manera el paralelismo y añade características que no existen en forma primitiva en Ada, como los operadores NAND y NOR, muy utilizados en electrónica. La sintaxis de este lenguaje es similar a la de Ada y eso es porque quienes diseñaron el lenguaje, trabajaban en Ada en un principio. Este es un caso de especialización de lenguaje para acercarlo al dominio.

Por las razones previamente mencionadas, es posible afirmar que en algunos contextos y dominios, el uso de lenguajes de propósito general no hubiese sido lo más apropiado, existiendo para cada caso una mejor alternativa tomada con un lenguaje específico del dominio.

3. RESULTADOS

La línea de investigación descripta en este artículo es reciente y por lo tanto los resultados obtenidos se relacionan con la descripción de los de los pasos que se deben realizar para construir un DSL.
El proceso de desarrollo de lenguajes específicos del dominio es similar a otros procesos de desarrollo de software. Por ello genéricamente se pueden encontrar las fases de análisis, diseño e implementación. Además también conviene analizar una fase previa a éstas: La decisión. A continuación se describen cada uno de los pasos mencionados previamente.

Decisión: Si bien parece que está fase es única en este proceso de desarrollo, también se puede ver en los procesos de desarrollo de software convencionales, ya que implícitamente se está decidiendo si es conveniente empezar un desarrollo desde cero o mejorar o comprar un sistema existente. Para los lenguajes específicos del dominio esta primera fase es muy importante ya que al principio parece que construir un DSL es la mejor opción pero luego por falta de recursos el desarrollo no pasa de una librería de clases y métodos. Por ello antes de embarcarse en esta empresa es mejor saber si se cuenta con los recursos, como un analista de dominio, experiencia en el dominio y experiencia en creación de lenguajes. También saber si se va a crear un lenguaje desde cero o especializar alguno existente.

Análisis: Los programas escritos en un DSL solamente especifican una parte del comportamiento, ya que otra parte está implícita. Es por eso que se necesita saber cuál es la parte variable y cuál es la parte fija de un dominio. En ingeniería de software a menudo se asocia a dominio el concepto de una familia de sistemas. Así se puede definir el análisis del dominio como el proceso de identificar, analizar y representar un modelo del dominio y la arquitectura del software desde el estudio de sistemas ya existentes, la tecnología subyacente, la tecnología emergente, historias de desarrollo y otras cuestiones involucradas con el dominio. El análisis del dominio es una práctica relativamente nueva, a pesar de haber sido propuesta hace alrededor de 10 años, aún sigue siendo el tópico central de algunas investigaciones. En muchos casos el análisis del dominio se hace de una forma ad-hoc, de más esta decir que esta es una práctica no recomendable ya que existen técnicas y herramientas para hacerlo de forma disciplinada. Entre los varios métodos para llevar a cabo el análisis del dominio se destacan: ODM (Organization Domain Modeling), DSSA (Domain Specific Software Architectures) y FODA (Feature Oriented Domain Analisis). En síntesis se puede ver al análisis del dominio como un proceso cuya entrada es todo lo que tenga que ver con el dominio desde códigos fuentes hasta el consejo de los expertos, luego tiene una técnica que puede ser uno de estos métodos mencionados y por último la salida es modelo del dominio.

Diseño: Si bien no hay un lineamiento general para pasar del modelo del dominio hacia la implementación, van Deursen en [5] propone una metodología para armar diagramas de clases a partir de los diagramas de características (feature diagrams). El diseño especificado en UML se corresponderá a una familia de sistemas como una línea de producción. Antes de entrar en esta etapa se requiere primero hacer una serie de transformaciones sobre los diagramas de características para normalizarlos. Además van Deursen en [5] propone también un lenguaje escrito para los diagramas de características para facilitar su procesamiento y comunicación.

Implementación: Siguiendo la línea de van Deursen en [5] se puede tomar el diagrama de clases obtenido en el diseño para generar código en prácticamente cualquier lenguaje, o tomar la especificación del diagrama de características y transformarlo en una gramática BNF. En otras opciones de implementación también se puede construir una librería de clases y/o métodos que implementen la semántica y las operaciones descubiertas en el análisis y luego diseñar y construir un compilador o intérprete para traducir los programas hechos con el DSL. El compilador se puede construir desde cero o expandir uno existente. Construir desde cero tiene la desventaja que supone el desarrollo de

un proyecto grande y complejo pero también tiene la ventaja de que el compilador estará optimizado para dicho lenguaje, teniendo una mejor comunicación de errores, códigos más legibles ya que están en términos cercanos al dominio y también una mejor optimización. Adaptar o expandir un compilador existente tiene la ventaja que uno ya cuenta con un soporte inicial de operaciones que ya existen pero esto a veces limita la sintaxis haciendo que no quede claro en términos del dominio o teniendo mensajes de error poco claros. Las formas de extender un lenguaje son tres: (i) A través de un lenguaje embebido o de librerías específicas del dominio. La ventaja de esta forma es que se toma el lenguaje "como es" teniendo las características bases del mismo disponibles. La desventaja es que se puede perder expresividad ya que se debe trabajar con la sintaxis del lenguaje. También se puede extender a través de (ii) Macros o pre-procesamiento. Esta es una forma simple de extender un compilador pero produce dos problemas, no hay chequeo estático ni optimización a nivel del dominio y los errores no sólo no estarán expresados en términos del dominio sino que quizás sólo surjan en tiempo de ejecución. Por último se puede utilizar (iii) Compiladores o intérpretes extensibles. Ésta es una forma similar a la anterior pero la fase de pre-procesamiento está integrada al compilador. La ventaja es que se puede hacer un chequeo estático, mejor manejo de errores y una mejor optimización.

Como trabajo futuro a corto plazo se pretende construir un DSL para la enseñanza de la programación para alumnos de los últimos años de la escuela secundaria, con el propósito de facilitar la preparación previa necesaria para el ingreso a la universidad. Además del objetivo previamente mencionado, se pretende analizar otros dominios de aplicación, donde se necesite un DSL. Una vez identificado dicho dominio se procederá a la definición del DSL correspondiente.

4. FORMACIÓN DE RECURSOS HUMANOS

Las tareas realizadas en el contexto de la presente línea de investigación están siendo desarrolladas como parte de trabajos de Licenciatura en Ciencias de la Computación. En el futuro se piensa generar diferentes tesis de maestría y doctorado, como así también trabajos finales de especialización en ingeniería de software (dirigidas por docentes de la Universidad Nacional de San Luis y la Universidade do Minho) a partir de los resultados obtenidos en las tesis de licenciatura en curso.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] P. Hudak. Building domain-specific embedded languages. ACM Computing Surveys, 28(4es), December 1996.

[2] M. Mernik, J. Heering, T. Sloane. When and How to Develop Domain-Specific Languages. ACM Computing Surveys, 37(4es), December 2005.

[3] J. Sanchez Cuadrado, J. García Molina. A Model-Based Approach to Families of Embedded Domain-Specific Languages. IEEE Transactions on Software Engineering, vol 35 Nº 6, November/December 2009.

[4] A. van Deursen, P. Klint, J. Visser. Domain-Specific Languages: An Annotated Bibliography. ACM Sigplan Notices, Vol. 35, No. 6, 2000.

[5] A. van Deursen, P. Klint. Domain-Specific Language Design Requires Feature Description. CIT Journal of Computing and Information Technology, Special Issue on Domain-Specific Languages, Part II, Eds. R. Laemmel, M. Mernik, Vol. 10, No. 1, pages 1-17, 2002.

[6] T. Kosar, P. Martinez, P. Barrientos, M. Mernik. A preliminary study on various implementation approaches of domain-specific languages. Inf. Softw. Technol., Vol. 50, No. 5, 2008.

[7] D. Wile. Lessons learned from real DSL experiments. Science of Computer Programming, Vol. 51, No. 3, 2004.