Uso de Lenguajes Específicos del Dominio en Enseñanza de Programación

Mariano Luzza[(1)], Mario Berón[(1)], Germán Montejano[(1)]
[(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: { mluzza, mberon, gmonte}@unsl.edu.ar

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.
Un Lenguaje Específico del Dominio (LED) es un conjunto reducido de construcciones y operaciones que brindan una mayor expresividad y optimización para un dominio particular.
Un área de especial aplicación para los LED es en el ámbito de la enseñanza en programación, ya que posibilitan abstraerse de los problemas particulares de los Lenguajes de Propósito General (LPG) para centrarse en el tema particular que se desea enseñar.
En este artículo se presenta una línea de investigación que tiene por objetivo descubrir y proponer mejoras en la enseñanza de programación a través del uso de LED.

Palabras Claves: Lenguaje Específico del Dominio, Enseñanza de Programación, Generador de Aplicaciones.

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.

1. INTRODUCCIÓN

Hoy en día es muy común encontrar computadoras cada vez más a nuestro alrededor. No sólo en la forma tradicional de computadoras de escritorio o notebooks, sino que también en celulares, vehículos, electrodomésticos, etc. Todas y cada una de esas computadoras utilizan software para cumplir sus acometidos. Y esta vertiginosa aceleración en la demanda de software, necesita muchos más profesionales en computación de los que se producen habitualmente. El problema con esto radica en que muchas personas no aprenden fácilmente esta ciencia y se dan por vencidos en forma temprana. Parte de la dificultad se presenta en que los lenguajes de programación son muchas veces complejos sin necesidad, requiriendo por ejemplo aprender una forma de escribir, que no es natural para el principiante. En otras palabras, se requiere el aprendizaje de una nueva sintaxis, la sintaxis del lenguaje de programación. Otra debilidad se presenta a la hora de mostrar resultados al usuario. Los alumnos invierten gran cantidad de tiempo en la creación del programa, que al ser ejecutado, resulta ser un monótono programa de texto. Esta experiencia puede resultar aun peor si ocurren errores de sintaxis. Ciertamente los principios de la programación y la lógica no se van a convertir en más sencillos, por lo menos no en este trabajo. Lo que sí es posible cambiar y es el foco de esta investigación, es que los lenguajes y sus herramientas de trabajo sean más sencillas de comprender, de utilizar y con resultados más atractivos. Es por ello que se necesitan más y mejoras métodos y herramientas para la enseñanza de programación.

Este artículo está organizado como sigue. La sección 2 presenta la línea de investigación y tiene cuatro finalidades importantes. La primera explica qué es un LED, la segunda menciona los usos comunes de los LED, la tercera muestra ejemplos de LED en educación y la cuarta presenta el objetivo de la línea de investigación. La sección 3 describe 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 (LED) es un conjunto reducido de construcciones y operaciones que brindan una mayor expresividad y optimización para un dominio particular. Según [HUDAK96] un LED es "la última abstracción", que captura precisamente la semántica de un dominio de aplicación. Algunos LED 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 sucede lo mismo cuando se desea describir soluciones fuera de su dominio. Algunas industrias poseen también sus propios LED. Por ejemplo, un estudio contable que se especializa en pago de impuestos podría diseñar un LED con la finalidad de generar parte de los formularios de impuestos de cada cliente. También sería útil para un equipo de control de calidad, contar con un LED para generar los procesos de una empresa y los documentos asociados. Otras áreas donde también se podrían usar LED incluyen diagramas de conexión de componentes electrónicos, armado de dietas o rutinas, entre otras.

2.2. Utilización

Los usuarios de los LED crean modelos que luego se compilan o se traducen a otros artefactos. Es común hallar LED 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 LED, se pueden especificar plantillas que lean un modelo del LED 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 LED son creados cuando un grupo de usuarios (no necesariamente desarrolladores) tienen que generar código similar para varios productos. Por ejemplo, sería útil para un equipo de desarrollo de sitios web, contar con un LED para generar la navegabilidad de las diferentes páginas y que cree tanto un esqueleto del sitio como una herramienta que verifique dicha navegabilidad. El principal beneficio de los LED, es que pueden ser más sencillos de entender por los usuarios, tanto del LED como quizás también de los que usarán el producto generado por el LED, 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 LED, la representación es más sencilla, sólo se describe cómo resolver el problema en términos del dominio y no hay construcciones especiales de código para poder completar el algoritmo como ocurriría en un LPG. Gracias a esto, se vuelve 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 LED, ya que estará en términos del dominio y que cualquier usuario del entorno debería manejar. Aun así, 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

A continuación se describen brevemente un conjunto de LEDs usados en educación.

2.3.1. Logo

Diseñado en el MIT (Instituto Tecnológico de Massachussets) como un lenguaje de aprendizaje, Logo es un lenguaje de computadora completo derivado de LISP. Su principal característica es que es un lenguaje para aprender. Es una herramienta útil para enseñar el proceso de aprendizaje y de pensamiento. En Logo, el alumno da comandos para dirigir a una tortuga con

instrucciones simples como avanzar y girar. Logo no está limitado a un tópico en particular o a una materia específica. Sin embargo, es más comúnmente utilizado para la exploración de las Matemáticas ya que al moverse, la tortuga de Logo dibuja gráficos naturalmente matemáticos. Debido a que la tortuga se mueve una determinada distancia y gira un número dado de grados, el estudio de geometría mediante la construcción e investigación de polígonos y figuras hace de Logo una herramienta de aprendizaje poderosa a la vez que es entretenida.

2.3.2. Scratch

Fue desarrollado por "el grupo 'jardín de infancia' para toda la vida" en el Media Lab del MIT por un equipo dirigido por Mitchel Resnick. Scratch es un entorno de aprendizaje de lenguaje de programación, que permite a los principiantes aprender a escribir de manera sintácticamente correcta primero. El entorno se usa principalmente para armar animaciones y juegos básicos. A su vez, permite investigar, introducirse y jugar con la programación de computadoras utilizando una interfaz gráfica muy sencilla y un lenguaje visual. El lema de Scratch es “imagina, programa y comparte”.

2.3.3. Alice

Alice fue desarrollado por los investigadores de la Universidad Carnegie Mellon. Es un lenguaje de programación educativo libre y abierto. Al igual que Scratch, permite al alumno crear animaciones, que incluyendo la interacción con el usuario, se convierten en animaciones interactivas o videojuegos. Una evolución sobre Scratch es que presenta un mundo en tres dimensiones. Si bien sigue siendo una herramienta sencilla, es más compleja que Scratch.

2.3.4. Proyect Hoshimi

El Proyecto Hoshimi, es un juego en el que unos nanobots deben curar el cuerpo humano. Estos nanobots tienen una inteligencia artificial que es programada por el alumno. El juego posee originalmente, dos alternativas para el desarrollo del programa que se encargará de la inteligencia artificial: un lenguaje con un entorno sencillo pero que carece de potencial y flexibilidad o un lenguaje y un ambiente de desarrollo de software complejo, con potencial y flexibilidad pero más complejo de utilizar. En este contexto se desarrolló PH-Asistido [12], una herramienta híbrida entre las dos modalidades anteriores, que combina las ventajas de ambas.

2.4. Objetivo

Con lo descripto anteriormente, esta línea de investigación tiene como objetivo estudiar LED en general, sus procesos de desarrollo y su uso, especialmente en la enseñanza de programación. Por supuesto, también será de interés indagar en temas relacionados a la pedagogía.

3. RESULTADOS

Una de las principales tareas desarrolladas en el contexto de la línea de investigación descripta en este artículo es el ambiente de desarrollo de software (ADS) PH-Asistido. Este ADS permite desarrollar aplicaciones para el juego Proyecto Hoshimi (PH). En este juego, el usuario debe desarrollar una estrategia para la inteligencia artificial de unos nanobots que curan el cuerpo humano. El atractivo del juego, junto con los conceptos que maneja, lo hacían ideal para la enseñanza de la programación, especialmente a alumnos sin conocimientos previos. El problema radicaba en que originalmente, PH presentaba dos formas de desarrollar las estrategias:

1. Empleando el editor visual integrado en la plataforma. La ventaja de este modo es que el lenguaje es sencillo y está acotado al dominio (de hecho es un LED). La desventaja es la escasa expresividad y funcionalidades disponibles, como el uso de variables, muy necesarias para que los nanobots guarden su estado.
2. Usando algún IDE de .Net. La ventaja de este modo es que están accesibles todas los funcionalidades de la API del juego y todas las herramientas de un LPG (por ejemplo C# o VisualBasic.Net) como definición de tipos, uso de estructuras como listas y diccionarios y lo más importante, uso y definición de variables. Como contrapartida, se puede decir que los

alumnos deben aprender a programar con lenguajes orientados a objetos y lidiar con errores que entorpecen el aprendizaje como omitir palabras claves o símbolos de puntuación, problemas con los identificadores (escribirlos mal o no respetar las mayúsculas), respetar estructuras del código, por nombrar algunas dificultades.

Teniendo en cuenta esto, fue desarrollado PH-Asistido, que posee un editor visual para PH que parte de la idea del editor incorporado en la plataforma de PH (ítem 1), y además provee un potencial de trabajo similar al mencionado en el ítem 2.

En otra tarea llevada a cabo en el marco de esta línea de investigación, se construyó una API para desarrollar agentes de inteligencia artificial para el juego "Grab them by the Eyes". Este juego, desarrollado por Terry Cavanagh, propone al jugador situarse en el rol de un vendedor de comida ambulante que de repente encuentra competencia debido a otro vendedor que se instala cerca de su puesto. El problema con esta nueva competencia es que utiliza carteles luminosos en su puesto para atraer a los clientes. Así, el jugador debe crear carteles llamativos, para atraer más clientes que la competencia. El juego transcurre durante un periodo de seis días, donde cada día tiene tres etapas:

1. Compra de componentes de carteles.
2. Armado de carteles.
3. Resultado de los carteles.

Durante la primera etapa ambos vendedores se turnan uno a otro para comprar componentes de carteles. Además, cada día se sucede el vendedor que comienza la compra para dar equilibrio a esta etapa. La tienda de carteles presenta cinco partes de carteles en vidriera de diferentes tipos, puntajes y precios. Los tipos son: (i) mensaje; (ii) color; (iii) borde; (iv) efecto; y (v) marco. Los puntajes son un número natural que representa la cantidad de clientes que atraerá. El precio está determinado por la posición en la tienda, siendo el costo del primer cartel 10$, el segundo 20$ y así siguiendo. Cuando un vendedor compra una parte de cartel, se quita de la tienda y las demás se corren para ocupar la posición desierta, bajando el costo de esos componentes. Por ejemplo sean A, B, C, D y E partes de carteles en las posiciones 1 a la 5, si el jugador compra la parte C a 30$, luego D y E se correrán una posición, costando ahora 30$ y 40$ respectivamente. Además, la tienda suele tener más de cinco partes de carteles, siendo las sobrantes una reserva que no se muestra ni se puede comprar, pero que van apareciendo a medida que se compran las que están en vidriera. Cada turno, un jugador puede comprar un único componente. Los turnos se suceden hasta que ambos jugadores se queden sin dinero para comprar o se acaben los componentes.

En la segunda etapa los vendedores arman sus carteles. Los carteles están divididos en marcos, que al principio de la partida son dos, pero se pueden comprar más en la tienda. Cada marco puede tener a lo sumo uno de cada tipo de componente restante (mensaje, color, borde y efecto). La suma de los puntos de los componentes usados es el puntaje total del cartel.

Al final, en la tercera etapa se computan los puntos (que representan clientes) en cada vendedor y los componentes utilizados pierden un punto hasta un mínimo de uno. Cuando transcurren los seis días, se suman todos los días y se define el ganador.

El desarrollo original de Terry Cavanagh es un juego donde un vendedor lo dirige en tiempo real el jugador y el otro es una inteligencia artificial (IA). Como trabajo de esta línea de investigación, se replicó el juego, pero donde ambos jugadores son IA, que se deben desarrollar a partir de una API que incluye el nuevo juego desarrollado. Esta API presenta las siguientes clases:

- Carta: representa cada parte de cartel. Tiene propiedades para saber el tipo y el puntaje.
- Cartel: es la composición de las partes.
- Jugador: es la clase abstracta que debe implementar quien desarrolla una IA.
- Partida: es el motor de la partida y presenta métodos públicos para que el jugador interactúe con el juego.

La clase "*Jugador*" presenta un único método abstracto denominado "*Jugar*". Es aquí donde el jugador implementa la inteligencia artificial. Dentro del método, recibe como parámetro un objeto del tipo "*Partida*" al que podrá interrogar para conocer la etapa del juego, las partes que tienen ambos jugadores, las partes de la tienda, etc. También posee métodos para interactuar, como comprar una parte y armar un cartel.

De esta forma, el alumno y el docente pueden utilizar este juego para practicar conceptos, tanto avanzados como inteligencia artificial pero también tópicos más simples como herencia y colecciones. Como trabajo futuro a corto plazo se pretende añadir otras funcionalidades, como por ejemplo nuevas reglas al juego que permitan analizar otros conceptos y también una consola de registro que permita depurar los programas realizados. Además, se pretende reutilizar la estrategia empleada en el desarrollo de esta API para crear otras herramientas para otras tareas que realizan los alumnos que no poseen editores o simplemente los resultados no son lo suficientemente atractivos. También se pretende crear un ambiente de desarrollo con un LED completo que utilice la API desarrollada para facilitar aún más el aprendizaje de los conceptos.

4. FORMACIÓN DE RECURSOS HUMANOS

Las tareas llevadas a cabo en esta línea de investigación están siendo realizadas en diferentes tesis correspondientes a la Licenciatura en Ciencias de la Computación. Se proyecta a corto plazo la continuación de esta investigación con la elaboración de nuevas tesis de licenciatura, tesis de maestría y doctorado.

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, 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.

[8] F. P. Andrés, J. de Lara, and E. Guerra. Domain Specific Languages with Graphical and Textual Views. In Andy Schürr, Manfred Nagl, and Albert Zündorf, editors, AGTIVE, volume 5088 of Lecture Notes in Computer Science, p. 82–97, 2007.

[9] F. Arefi, C.E. Hughes, D.A. Workman. The object-oriented design of a visual syntax-directed editor generator. In Computer Software and Applications Conference, 1989. COMPSAC 89., Proceedings of the 13° Annual International, p. 389–396, 1989.

[10] Farah Arefi, C. E. Hughes, D. A. Workman. Automatically generating visual syntax directed editors. Commun. ACM, 33:349–360, March 1990.

[11] Stavroula Georgantaki, S. Retalis. Using educational tools for teaching object oriented design and programming. Journal of Information Technology Impact (Jiti), 7(2):111–130, 2007.

[12] M. Luzza, M. Berón, M. Peralta, C. Salgado. PH-Helper: a Syntax-Directed Editor for Hoshimi Programming Language. In X Information Technology Applied to Education Workshop, Computer Science & Technology Series. XVIII CACIC Selected Papers, 2013.