Propuesta de Trabajo de Inserción Profesional

Título:

Desarrollo de un Entorno Integrado de Aprendizaje de Programación utilizando Editores Proyetivos siguiendo la didáctica de Gobstones

Alumno:
Ariel Alvarez

Director:
Ing. Nicolás Passerini

Codirector:
Ing. Javier Fernándes

Carrera:
Tecnicatura Universitaria en Programación Informática

Propuesta de Trabajo de Inserción Profesional

12 de octubre de 2015

Resumen

El lenguaje Gobstones posee una secuencia didáctica bien definida que ha demostrado ser eficaz, pero el entorno de programación usado actualmente, aún cumpliendo las funciones de edición para las cuales fue creado, no refuerza activamente esta secuencia didáctica ni suple las necesidades de una comunidad creciente de usuarios. El presente trabajo busca desarrollar un *ILE* sobre un *Editor Proyectivo*, haciendo uso de sus cualidades intrínsecas para facilitarle al alumno la comunicación de soluciones en términos de conceptos en lugar de trabajar sobre texto plano.

1. Introducción

En esta sección se detallan brevemente las características del lenguaje Gobstones y de los editores proyectivos. Luego se presenta un caso particular de ambiente de desarrollo de lenguajes basados en tecnología proyectiva, que será utilizado para implementar el entorno propuesto.

1.1. El lenguaje Gobstones

Citando al sitio oficial de Gobstones[3]:

> Gobstones es un lenguaje de programación ideado por Pablo E. Martínez López y Eduardo A. Bonelli en el año 2008 pensando para satisfacer las necesidades de los alumnos de la reciente carrera Tecnicatura Universitaria en Programación Informática que se dicta en la Universidad Nacional de Quilmes. Este lenguaje de programación fue pensado en el marco de un primer año de una carrera informática para aquellas personas que no tienen conocimientos previos de programación, basándose en tres elementos fundamentales que forman parte de la filosofía de Gobstones: el desarrollo de la abstracción, el aprendizaje de la división de tareas y la obtención de pautas de estilo que favorezcan la posterior interpretación de los programas generados, todo manteniendo un alto grado de simplicidad conceptual y eliminando elementos superfluos que claramente complican el aprendizaje inicial

1.2. Entorno Integrado de Aprendizaje

Los entornos integrados de aprendizaje[2] cubren un gran número de herramientas. En el presente trabajo al hablar de *Entorno Integrado de Aprendizaje de Programación*, de ahora en más *ILE* (*Integrated Learning Environment*), nos referiremos a una aplicación diseñada para la enseñaza de la programación, orientada a cumplir cierta estrategia didáctica y facilitar la creación de planes de estudio por parte del docente.

Es importante diferenciar al *ILE* de un *IDE* (*Integrated Development Environment*) ya que este último se enfoca en el desarrollo de programas, y por lo tanto posee funcionalidades que en ciertos contextos de aprendizaje pueden resultar perniciosas. Por ejemplo, es esperable que un *IDE* provea mecanismos de depuración de código inspeccionando el estado del programa en una ejecución secuencial. Sin embargo, en un *ILE* esto puede ser considerado pernicioso dentro de ciertas estrategias didácticas, ya que permite al alumno buscar soluciones mediante fuerza bruta en lugar de utilizar las herramientas abstractas que se intentan transmitir.

1.3. Editores Proyectivos

El término Editor Proyectivo fue acuñado por Martin Fowler en el año 2005[5], al intentar plantear un ambiente de desarrollo donde el programador pueda expresar sus ideas en términos de conceptos en lugar de texto. Lo que vemos como texto pasaría entonces a constituir una representación editable del concepto al que hace referencia (y al cual Fowler llama *representación abstracta*).

De esta manera, los conceptos del lenguaje son el dominio de los editores proyectivos, y decimos que un programa es una *representación abstracta* construida utilizando dichos conceptos. Para modificar esta representación el programador interactúa con una interfaz de usuario, llamada *representación editable*, sobre la cual la *representación abstracta* se *proyecta* en forma de texto[4].

A su vez, la *representación abstracta* puede persistirse de diferentes maneras a diferentes soportes, con lo cual se introduce la idea de *representación persistida* para hablar del formato en que guardará el programa, ya sea en una base de datos, un archivo binario, un texto con formato XML, etc.

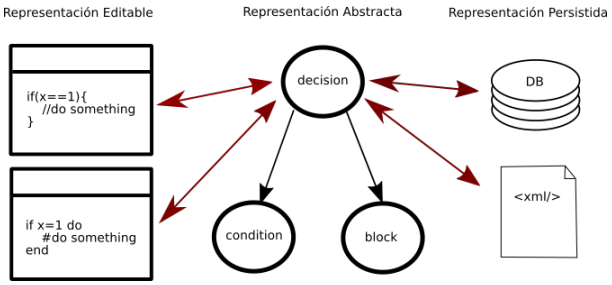

Figura 1: Representaciones de un programa en un editor proyectivo

Esto tiene varias consecuencias:

- Deja de necesitarse un parser para el lenguaje, volviéndolo más sencillo de extender.
- El programador trabaja más cerca de los conceptos que quiere expresar.
- Reduce drásticamente los posibles errores de sintaxis.
- El editor trabaja directamente con las instancias de los conceptos, con lo cual:
  - es más sencillo analizar el programa
  - se simplifica la construcción de herramientas (ej: refactors, migrado de versiones de lenguaje, intentions, etc)
  - se mejora la performance del editor al eliminarse la etapa de parseo.

1.4. MPS Workbench

MPS (Meta programming System)[8] es un ambiente de trabajo orientado al desarrollo de lenguajes específicos de dominio, de la empresa JetBrains, basado en las ideas de Fowler; hoy en día se posiciona como uno de los referentes más importantes en editores proyectivos[1]. Entre los productos desarrollados utilizando MPS se encuentran: *MetaR* [9], un ambiente especializado en biomedicina computacional; *mbeddr* [7], un ambiente orientado a software embebido.

2. Motivación

En el libro *"Las bases conceptuales de la Programación. Una nueva forma de aprender a programar"*[3] se presenta una secuencia didáctica bien detallada junto con el entorno *PyGobstones*[6]. Sin embargo notamos que, en el marco de una adopción cada vez mayor de esta secuencia didáctica, aparecen nuevas necesidades que el entorno actual no cubre o cubre solo parcialmente:

- Disponer de un ambiente que sea facilmente extensible y posea una arquitectura que agilice la colaboración de la comunidad de Gobstones.
- Utilizar tecnologías que posean un buen soporte.
- Que el entorno sea sencillo de instalar.

- Que el entorno constituya un refuerzo de la se secuencia didáctica, y no sólo una herramienta de edición.
- Que se encuentre cosntruido sobre soluciones probadas y usadas, en lugar de implementar todos sus componentes desde cero.

3. Propuesta

Se propone desarrollar un *ILE* centrado en el lenguaje Gobstones. Para ello se utilizará *MPS Workbench*, cubriendo las necesidades mencionadas en la sección anterior:

- soporta tanto extensión de lenguaje como extensión del entorno mediante la creación de plugins.
- los features de lenguaje pueden ser activados y desactivados durante el uso del entorno para cumplir con la secuencia didáctica.
- es soportado por JetBrains, uno de los líderes mundiales en desarrollo de entornos de programación.
- el entorno sólo necesita Java Runtime Environment (siendo que se trabajará sobre la implementación de un intérprete Gobstones, con lo cual se vuelve innecesario contar con un compilador).

Entendemos que naturaleza proyectiva de *MPS* cataliza ciertas características deseables en un *ILE* :

- permite al alumno concentrarse en el concepto que desea expresar al reducir notablemente los posibles errores de sintaxis.
- el programa siempre se presenta al alumno con un formato estándar (bien indentado, secciones organizadas, etc), ayudándolo a incorporar más rápidamente estas buenas prácticas al reconocerlas como algo natural. En otras palabras, el alumno no posee de antemano una noción de estilos *correctos*, sino que se construye a medida que es expuesto a estos.

En particular, de desea que el *ILE* posea las siguientes características:

- una implementación de Gobstones
- que incluya la idea de proyecto y ejercicio como mecanismos de organización de código.
- que permita limitar los features de lenguaje que pueden ser usados para resolver cierto ejercicio.
- que reconozca la especificación del propósito y precondición de funciones y procedimientos (que no sean un comentario más en el código).
- que el alumno programe sobre un editor proyectivo
- que provea autocompletado, atajos de teclado y navegación de código entre otras funcionalidades comunmente encontradas en editores (siempre y cuando no vayan en contra de la estrategia didáctica)
- que provea un mecanismo de integración con otras herramientas que puedan desarrollarse en el futuro, mediante la creación de plugins.

4. Plan de trabajo

Se organiza el desarrollo en sprints con duración de dos semanas.

- Sprint 1 Modelo puro de Gobstones. Editor que soporte comandos básicos (*Poner*, *Sacar*, *Mover*) y procedimientos sin parámetros.
- Sprint 2 Que el ditor soporte procedimientos con parámetros, variables, ciclos, expresiones booleanas y aritméticas.

- Sprint 3 Se comienza el intérprete del lenguaje. Generación de un tablero a partir de comandos básicos.
- Sprint 4 Soporte a nivel lenguaje para los tableros iniciales. Se concluye el intérprete del lenguaje.
- Sprint 5 Subproyecto de deploy, generación del entorno. Se agrega soporte para estructura de proyecto y ejercicio.
- Sprint 6 Ejercicios configurables con nivel de lenguaje.

Referencias

[1] Markus Voelter y col. «Language modularity with the MPS language workbench». En: *Software Engineering (ICSE), 2012 34th International Conference* (1905). DOI: 10.1109/ICSE.2012.6227070.

[2] Peter Brusilovsky. «Intelligent learning environments for programming: The case for integration and adaptation». En: *J. Greer (ed.) Proceedings of AI-ED'95, 7th World Conference on Artificial Intelligence in Education*. International Center for Scientific y Technical Information, Moscow, Russia, 1995, págs. 1-8.

[3] Pablo E. Martínez López. *Las bases conceptuales de la Programación: Una nueva forma de aprender a programar*. La Plata, Buenos Aires, Argentina, 2013. ISBN: 978-987-33-4081-9. URL: http://www.gobstones.org/bibliografia/Libros/BasesConceptualesProg.pdf.

[4] Markus Voelter y col. «Towards User-Friendly Projectional Editors». En: *7th International Conference on Software Language Engineering (SLE)*. 2014.

[5] M. Fowler. *Language Workbenches: The Killer-App for Domain Specific Languages?* URL: http://martinfowler.com/articles/languageWorkbench.html.

[6] Gobstones. *Qué es Gobstones*. URL: http://www.gobstones.org/?page_id=50.

[7] itemis y fortiss. *mbeddr*. URL: http://mbeddr.com/.

[8] JetBrains. *MPW Workbench*. URL: https://www.jetbrains.com/mps/.

[9] Campagne Laboratory. *MetaR*. URL: http://metaR.campagnelab.org.