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 Fernandes

Carrera:
Tecnicatura Universitaria en Programación Informática

Propuesta de Trabajo de Inserción Profesional [1]

***Resumen*—El lenguaje Gobstones posee una secuencia didáctica bien definida que ha demostrado ser eficaz tanto en cursos iniciales universitarios como en escuelas secundarias. En el marco de una comunidad creciente de usuarios, proponemos el desarrollo de un *Entorno Integrado de Aprendizaje de Programación* a partir de una implementación de Gobstones 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, reduciendo así elementos superfluos que pudieran entorpecer la secuencia didáctica.**

I. Introducción

En la República Argentina la enseñanza de la programación en el segundo ciclo primario y primer ciclo secundario se plantea utilizando lenguajes eminentemente visuales [4]. Esto permite a los alumnos concentrarse en aquello que desean expresar (es decir, el programa que pretenden crear) al eliminar ciertas dificultades inherentes en lenguajes sobre soporte de texto. Por ejemplo, el lenguaje Scratch[7] únicamente permite construcciones *sintácticamente válidas* ya que cada comando del lenguaje es conformado por bloques visuales encastrables, de tal manera que dos bloques solo encastran cuando constituyen una combinación válida.

Luego, cuando el alumno pasa a un ciclo superior secundario o a la universidad, se le presentan lenguajes basados en texto, en los cuales los errores de sintaxis y de tipado son posibles. En particular en la Universidad Nacional de Quilmes, en la materia de Introducción a la Programación, se utiliza el lenguaje Gobstones[2], creado específicamente para la enseñanza de programación.

Si bien Gobstones cuenta con un modelo acotado y una secuencia didáctica clara, que lo convierten en una gran herramienta para la enseñanza universitaria y de ciclos superiores del secundario; al ser basado en texto presenta un nivel de complejidad que puede no resultar adecuado para el segundo ciclo de la educación primaria o el primer ciclo de la educación secundaria.

En este contexto surge la propuesta de crear una implementación de Gobstones que permita únicamente sintaxis válida, de manera similar a los lenguajes visuales ya usados en la enseñanza de la programación, reduciendo así elementos superfluos que pudieran entorpecer la secuencia didáctica planteada para el segundo ciclo de la educación primaria y el primer ciclo de la educación secundaria. Además, se presenta a esta implementación de Gobstones en un entorno que acompañe su secuencia didáctica, tanto desde la construcción de ejercicios y planteo de problemas, como la inclusión paulatina de conceptos nuevos durante el proceso de aprendizaje. Se pretende lograr una continuidad entre la enseñanza de la programación utilizando componentes visuales y texto, volviendo más gradual la transición entre uno y otro.

Para lograr esta experiencia cercana al texto pero con la ausencia de errores sintácticos, resulta idóneo el uso de un Editor Proyectivo, término 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*). 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[3].

De esta manera, el editor solamente permite ingresar construcciones sintácticamente válidas en un formato estandarizado (espacios, identación y demás elementos estéticos son dados por el editor, no por el usuario).

En la sección *II* se presenta la tecnología a usar y en la sección *III* se describe el desarrollo del modelo conceptual del lenguaje Gobstones en términos de esa tecnología. En la sección *IV* se muestra cómo este modelo conceptual se proyecta sobre el editor. Una vez creado el editor, se procede a implementar el intérprete del lenguaje y la renderización de los tableros inicial y final. Teniendo el lenguaje básico funcionando, se trabaja en la sección *V* sobre el sistema de inferencia de tipos, orientado a asistir al estudiante mediante mensajes de error legibles. En la sección *VI* se analizan problemas típicos de los editores proyectivos y se busca mejorar la experiencia de usuario aplicando diferentes técnicas que facilitan una edición más familiar, es decir, más cercana a una expriencia de edición de texto. Luego en la sección *VII* se extiende el proyecto agregando un *lenguaje de definición de ejercicios*, que constituye un lenguaje específico de dominio cuya finalidad es

permitirle al docente plantear ejercicios, desde título y descripción hasta restriciones de features de lenguaje y análisis de código. Por último en la sección *VIII* se cierra el informe con una conclusión e ideas sobre el camino que el proyecto pudiera seguir a futuro.

II. Tecnología proyectiva a usar

De los entornos proyectivos existentes hoy en día, se decide utilizar el workbench Meta Programming System (MPS)[10] de la empresa Jetbrains, en su versión 3.3. Se trata de un entorno orientado al desarrollo de lenguajes maduro y estable, sobre el cual se realizaron exitosamente diferentes proyectos, entre los cuales se cuentan:

- MetaR[9]: un IDE que utiliza el lenguaje R para facilitar el análisis de datos biológicos.
- mbeddr[8]: un IDE orientado a la programación sobre hardware, que extiende el lenguaje C y soporta verificación formal, máquinas de estado y variabilidad en líneas de productos, entre otros.
- YouTrack[11]: un gestor de proyectos.

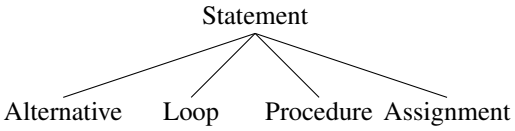
MPS brinda un DSL para la definición de conceptos puros del lenguaje a implementar, y sobre estos la posibilidad de describir cómo este modelo se renderizará, comportamiento específico para cada concepto, sistema de tipos, etc. Al ser todas estas incumbencias transversales a los conceptos, se organizan en forma de aspectos. Los conceptos se comportan de manera similar a una clase en programación orientada a objetos, en tanto y en cuanto admiten extensión por herencia e implementación de interfaces. A su vez, estas construcciones determinan las instancias de nodos que compondrán un programa, comparables a los nodos de un árbol de sintaxis abstracta.

A partir de los lenguajes definidos en esta herramienta es posible generar un IDE autónomo o pluguins para IDEs pre-existentes.

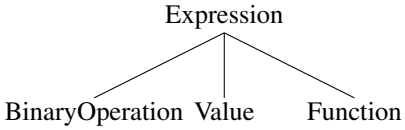
III. Modelo conceptual del lenguaje Gobstones

Se comienza modelando el lenguaje Gobstones en términos de conceptos. Como puede observarse en el ejemplo de *Fig.1*, para cada concepto pueden definirse sus posibles nodos hijo, propiedades y referencias a otros nodos. Se organizan en una jerarquía, pudiendo extender de otros conceptos e implementar interfaces. En este caso, puede verse que el concepto *IfElseStatement* extiende del concepto abstracto *Statement*, y sus posibles hijos son una expresión, que será la condición de la alternativa, un bloque de sentencias para el caso en que la condición sea verdadera, y otro bloque de sentencias para el caso contrario.

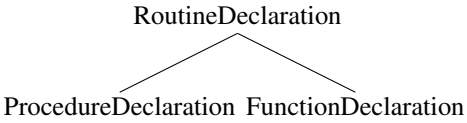
Los conceptos más importantes serán *Statement*, que denota un comando, y *Expression* que denota una expresión que puede ser evaluada. De manera simplificada, tenemos que el primer nivel de la jerarquía de sentencias queda dado por:



Y el primer nivel de la jerarquía de expresiones se compone de:



Además, se tiene una jerarquía separada para la definición de rutinas:



Decimos entonces que un programa gobstones básico se encuentra dado por una colección de sentencias y una colección de definición de rutinas.

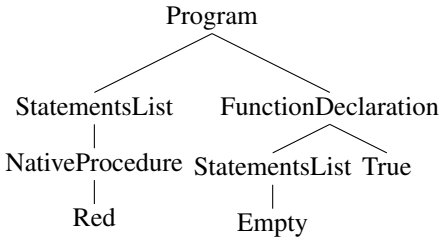
Donde el siguiente programa:

```
program {
  Poner(Rojo)
}

function verdadero(){
  return(True)
}
```

```
concept IfElseStatement extends    Statement
                        implements <none>

  instance can be root: false
  alias: if
  short description: Condicional

  properties:
  << ... >>

  children:
  condition    : Expression[1]
  ifTrueBlock  : StatementList[1]
  ifFalseBlock : StatementList[1]

  references:
  << ... >>
```

Figura 1. Definición del concepto para la alternativa condicional

Corresponde a un modelo conceptual con la estructura:



IV. Implementación del Editor

V. Implementación del Intérprete

V-A. *Edición del tablero inicial*

V-B. *Renderización del tablero final*

VI. Inferencia de tipos

VII. Mejorando la experiencia de Usuario

VII-A. *Edición de operaciones binarias*

VII-B. *Borrado de nodos*

VIII. Lenguaje de Definición de Ejercicios

IX. Conclusión

X. Anexo

X-A. *Configuración de los proyectos*

XI. Introducción

En esta sección se detallan brevemente las características del lenguaje Gobstones, se determina qué se entiende por *Entorno Integrado de Aprendizaje de Programación* y se da una introducción al funcionamiento de los editores proyectivos.

XI-A. *El lenguaje Gobstones*

Gobstones es un lenguaje de programación que fue diseñado para la enseñanza teniendo como elementos fundacionales de su filosofía: 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. Actualmente es soportado por diversas herramientas, entre las cuales se destaca el editor [6]

XI-B. *Entorno Integrado de Aprendizaje*

Los entornos integrados de aprendizaje[1] 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.

XI-C. *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[3].

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.

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.

XII. Motivación

En el libro *"Las bases conceptuales de la Programación. Una nueva forma de aprender a programar"*[2] se presenta una secuencia didáctica bien detallada junto con el entorno *PyGobstones*[6]. Notamos que, en el marco de una adopción cada vez mayor este lenguaje, puede resultar útil complementar las herramientas existentes con otras que refuercen activamente su secuencia didáctica, acercando al alumno a las abstracciones que se desean transmitir.

A su vez, se reconoce que los editores proyectivos poseen ciertas características deseables en una herramienta de enseñanza:

- 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.

XIII. Propuesta

Se propone desarrollar un *ILE* centrado en el lenguaje Gobstones a partir de un *Editor Proyectivo*:

- que incluya una implementación de Gobstones
- que incluya la idea de proyecto y ejercicio como mecanismos de organización de código.
- que permita limitar las herramientas de lenguaje que puedan ser usadas 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 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.

XIV. 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] 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.

[2] 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.

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

[4] Federico Sawady O'Connor Pablo Factorovich. *Actividades para aprender a Program.AR*. 2015. ISBN: 978-987-27416-1-7.

[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] Lifelong Kindergarten Group. *Scratch*. URL: https://scratch.mit.edu/.
[8] itemis y fortiss. *mbeddr*. URL: http://mbeddr.com/.
[9] Campagne Laboratory. *MetaR*. URL: http://metaR.campagnelab.org.
[10] JetBrains s.r.o. *MPW Workbench*. URL: https://www.jetbrains.com/mps/.
[11] Jetbrains s.r.o. *YouTrack*. URL: http://www.jetbrains.com/youtrack/.