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 [5]. 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[8][11] 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[6] 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[4].

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)[12] 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[10]: un IDE que utiliza el lenguaje R para facilitar el análisis de datos biológicos.
- mbeddr[9]: 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[13]: un gestor de proyectos.

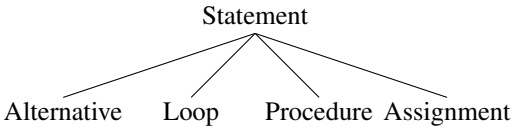
MPS brinda un DSL para la definición de conceptos puros del lenguaje a implementar, y sobre estos la posibilidad de describir cómo este modelo se renderizará, comportamiento específico para cada concepto, sistema de tipos, etc. Al ser todas estas incumbencias transversales a los conceptos, se organizan 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.

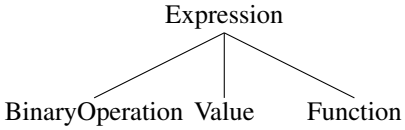
III. Modelo conceptual del lenguaje Gobstones

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.

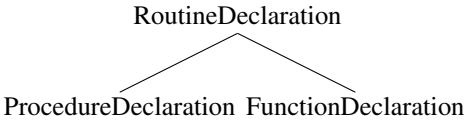
Los conceptos más importantes serán *Statement*, que denota un comando, y *Expression* que denota una expresión que puede ser evaluada. De manera simplificada, tenemos que el primer nivel de la jerarquía de sentencias queda dado por:



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.

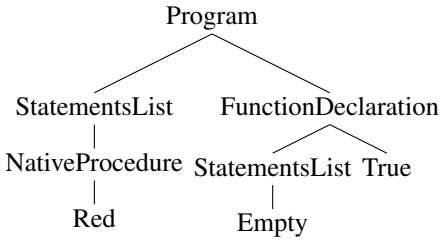
Donde el siguiente programa:

```
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

Una vez completado el modelo conceptual con los elementos del lenguaje Gobstones, según la especificación de *Las bases conceptuales de la Programación: Una nueva forma de aprender a programar*[2], se procede a definir el aspecto de editor para cada uno de ellos. Para ello, se hace uso de otro DSL provisto por MPS, que permite definir los layouts en los cuales los conceptos se renderizarán.

```
<default> editor for concept FunctionDeclaration
  node cell layout:
    [-
    # alias # { name } ( (- % arguments % /empty cell: <default> -) ) {
      (- % statements % /empty cell: -)
      return ( % return % )
    }
    /folded cell: [> # alias # { name } <]
    -]

  inspected cell layout:
    <choose cell model>
```

Figura 2. Definición del aspecto de edición para el concepto de declaración de función

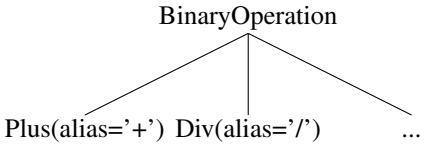
En la *Fig.2* se observa el aspecto de edición definido para la declaración de funciones. En este quedan mapeadas las propiedades e hijos del concepto *FunctionDeclaration*.

Esta definición provee un *binding bidireccional* entre el modelo conceptual y el editor, comparable con un patrón de arquitectura MVP [7][3]. Como consecuencia, cualquier cambio en el modelo se verá reflejado instantáneamente en el editor, y cualquier cambio realizado desde el editor modificará los nodos del modelo conceptual; habilitando también la posibilidad de tener diferentes vistas para un mismo modelo, cuya utilidad se verá en la sección siguiente.

```
<default> editor for concept BinaryOperation
  node cell layout:
    [- % left % # alias # % right % -]

  inspected cell layout:
    <choose cell model>
```

Figura 3. Definición del aspecto de edición para el concepto abstracto de operación binaria

El diseño del editor busca respetar los principios de la programación orientada a objetos. Un claro ejemplo son los subconceptos de *BinaryOperation* tales como los operadores lógicos y aritméticos, que reutilizan una misma vista. Para lograr esto se extrapola el clásico patrón *Template Method*[1], donde el aspecto de edición del concepto abstracto *BinaryOperation* define el layout de *Fig.3*, y cada subconcepto implementa un alias distinto, correspondiente con el símbolo de su operación.



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*

Referencias

[1] Erich Gamma y col. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0-201-63361-2.

[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] Microsoft. *Data Binding (WPF)*. 2014. URL: http://msdn.microsoft.com/en-us/library/ms750612(v=vs.110).aspx.

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

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

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

[7] Martin Fowler. *GUI Architectures*. URL: http://martinfowler.com/eaaDev/uiArchs.html.

[8] Lifelong Kindergarten Group. *Scratch*. URL: https://scratch.mit.edu/.

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

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

[11] John Maloney y col. «Scratch: A Sneak Preview». En: *Proceedings of the Second International Conference on Creating, Connecting and Collaborating through Computing, IEEE Computer Society*, págs. 104-109.

[12] JetBrains s.r.o. *MPW Workbench*. URL: https://www.jetbrains.com/mps/.

[13] Jetbrains s.r.o. *YouTrack*. URL: http://www.jetbrains.com/youtrack/.