PH-Asistido: Un Editor Dirigido por la Sintaxis para el Lenguaje Específico del Dominio Hoshimi

Mariano Luzza, Mario Berón, Mario Peralta, Carlos Salgado

Departamento de Informática - Facultad de Ciencias Físico-Matemáticas y Naturales
Universidad Nacional de San Luis
Ejército de los Andes 950 - San Luis - Argentina
{mluzza, mberon, mperalta, csalgado}@unsl.edu.ar

Resumen. Enseñar programación no es una tarea simple, porque el docente debe enfrentar los problemas tradicionales del proceso de enseñanza-aprendizaje, así como también sortear los inconvenientes proporcionados por el dominio de aplicación y por el lenguaje de programación.
Desde un punto de vista, es claro que si el dominio de aplicación no es atrayente, el estudiante no se sentirá motivado a resolver los problemas planteados por el docente. Por otra parte, los lenguajes de programación generalmente utilizan una sintaxis y un idioma que a los estudiantes les cuesta entender. Por estas razones, la elaboración de herramientas para facilitar la enseñanza de la programación representa un desafío relevante para los profesionales de la informática educativa.
Con el propósito de evitar los inconvenientes previamente expuestos, se desarrolló PH-Asistido, una herramienta cuyo principal propósito es facilitar la enseñanza de la programación a través de: i) La utilización de un dominio atrayente para la enseñanza de la programación: el Proyecto Hoshimi, ii) La extensión del Lenguaje del Proyecto Hoshimi con acciones que simplifican la programación, y iii) La definición e implementación de un editor visual proactivo dirigido por la sintaxis que evita que el estudiante disperse su atención en obstáculos poco relacionados con el problema a resolver.

Palabras Clave: LED, Editor Dirigido por la Sintaxis, Proyecto Hoshimi.

1 Introducción

Desde hace mucho tiempo, los científicos en computación han buscado como hacer más sencilla la tarea de enseñar a programar [14]. Por un lado, se investigó cuáles eran las causas de que sea tan difícil para el alumno aprender los conceptos básicos de programación. En este contexto, una de las fuentes principales de información fueron las asignaturas previas que ha tenido el alumno. Por otro lado, se diseñaron lenguajes más didácticos y cuyo principal objetivo es facilitar el aprendizaje de los conceptos de programación. Ejemplos de este tipo de lenguajes son *Logo* [5, 6, 7], *Scratch* [8, 9], *Lenguajes de Diseño*, entre otros tantos. Sin embargo, y a pesar de los esfuerzos

realizados por la comunidad científica, el problema no se ha resuelto, los alumnos no pueden asimilar de forma clara y simple los conceptos de programación. Sumado al inconveniente antes mencionado aparecen otros problemas que tienen que afrontar los estudiantes, como entender el enunciado del problema y escribir el programa para resolverlo. Es claro que ambos problemas pueden ser abordados con la guía y la didáctica del docente. No obstante, aparecen otros inconvenientes estrechamente vinculados a los lenguajes de programación como lo son el uso de palabras claves en inglés y sintaxis compleja (como por ejemplo los símbolos de puntuación correspondientes a las estructuras de sentencias). La dificultad relacionada con el idioma del lenguaje se puede evitar mediante el uso de lenguajes visuales. El obstáculo presentado por la sintaxis compleja se puede superar usando los Editores Dirigidos por la Sintaxis [10]. Estos editores son conscientes de la sintaxis o de las construcciones del lenguaje y por consiguiente proporcionan una ayuda significativa al principiante a la hora de expresar un algoritmo en un lenguaje de programación.

Otro problema que deben enfrentar los docentes de informática, es el enunciado del problema en sí mismo, esto engloba el contexto, el área de aplicación y los objetivos. Muchas veces los problemas propuestos no son lo suficientemente motivadores para los alumnos y/o los resultados no son "palpables". Es por eso que este tema (la elección de problemas interesantes y efectivos) también debe ser tenido en cuenta para lograr un proceso educativo exitoso.

El Proyecto Hoshimi [11], el cual será introducido en la sección 4, es otro intento de mitigar los problemas mencionados en los párrafos precedentes.

El artículo está organizado como sigue. La sección 2 y 3 introducen los conceptos de Lenguaje Específico del Dominio y Editor Dirigido por la Sintaxis respectivamente. La sección 4 presenta el Proyecto Hoshimi, un juego atractivo para la enseñanza de la programación. La sección 5 describe PH-Asistido un front-end para el Proyecto Hoshimi. La sección 6 expone un caso de estudio en donde se puede apreciar el uso de PH-Asistido para resolver problemas típicos de programación. Finalmente la sección 7 presenta las conclusiones de éste artículo.

2 Lenguaje Específico del Dominio

Un Lenguaje Específico del Dominio (LED) [1] es un conjunto reducido de construcciones y operaciones que brindan una mayor expresividad y optimización para un dominio particular. Según Hudak [2] un LED es la "última abstracción", la cual captura precisamente la semántica de un dominio de aplicación.

Algunos LED bien conocidos incluyen SQL y expresiones regulares entre otros. 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 intenta describir soluciones fuera de su dominio. Algunas industrias poseen también sus propios LED. 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, generalmente se necesita un lenguaje para describir reservas de vuelo. Otras áreas donde también se usan LED

incluyen el plan de rutas de navegación de una web, diagramas de conexión de componentes electrónicos, un árbol genealógico, etc.

Cuando se diseña un LED, se deben especificar claramente los conceptos y relaciones relevantes para el dominio [4, 12] al cual dicho lenguaje va a estar destinado, como por ejemplo un árbol genealógico. Para armar un árbol genealógico es necesario representar las relaciones familiares entre distintas personas. De aquí se puede rescatar como conceptos importantes "Persona" y "Parentesco".

Los usuarios de un LED crean modelos los cuales son las instancias del LED [13, 17, 18]. Siguiendo el ejemplo del árbol genealógico, un modelo podría describir varias generaciones de una persona en particular o sólo sus relaciones familiares en la actualidad.

La ventaja de los LED es que al focalizarse en el dominio, los programas quedan descriptos en términos que manejan los usuarios del dominio [3]. De esta forma son los usuarios finales quienes construyen los programas, sin que sean necesariamente programadores. En este artículo se utilizará y extenderá el LED del Proyecto Hoshimi, un juego interesante para la enseñanza de la programación.

3 Editores Dirigidos por la Sintaxis

Existen dos tipos de editores dirigidos por la sintaxis [10, 19], los *reactivos* y los *proactivos*. En los *reactivos*, el editor conoce la sintaxis pero deja libre al programador para que escriba lo que desea. Mientras el programador escribe o después de realizar dicha tarea, el editor remarca de alguna forma las palabras clave y las estructuras de código. *WinEdt*, un poderoso editor de documentos [*La*]*Tex* para *Windows* es un ejemplo de este tipo de editores. WinEdt tiene esquemas de remarcación que se pueden personalizar para diferentes modos y un corrector ortográfico para varios lenguajes. Otra característica que suelen ofrecer los editores reactivos es el *autocompletado*. El *autocompletado* es la habilidad de completar palabras mientras se las escribe, de acuerdo al contexto. Esta característica mejora notablemente la productividad y al mismo tiempo reduce los errores. Hoy en día muchos editores traen implementado el *autocompletado*, uno de los más conocidos es el Intellisense de Microsoft. Intellisense además de conocer la sintaxis del lenguaje, también es sensitivo a la semántica y reconoce todas las variables declaradas hasta el momento de su uso.

Los editores *proactivos*, son más interesantes ya que además de conocer la sintaxis, proveen una guía para el desarrollador mientras escribe. En cada momento de la escritura de un programa, el editor conoce las construcciones que son aplicables y que se puede escribir. De esta manera, ayuda al programador sugiriendo la variable o el valor que se está buscando entre otras tantas funcionalidades. Claramente, el editor provee mucha ayuda, sobre todo para quien no conoce del todo la sintaxis del lenguaje. Es importante destacar que este tipo de herramientas pueden ser generadas automáticamente a partir de la gramática.

Es importante mencionar que la herramienta que se describe en la sección 5 es un editor *proactivo*.

4 Proyecto Hoshimi

El Proyecto Hoshimi fue creado en 2005 por Richard Clark para la *Imagine Cup*, una competencia de Microsoft. El Proyecto Hoshimi es un juego basado en la tecnología .Net, particularmente, en el lenguaje orientado a objeto C#. La idea básica consiste en programar unos robots diminutos (*NanoBots*) para curar enfermedades que afectan al tejido humando (el escenario del juego). La tarea de los alumnos consiste en programar las estrategias de los diferentes *NanoBots*, a través de un conjunto de sencillas e intuitivas acciones. Claramente, este juego resulta atractivo y a la vez suficientemente desafiante.

El Proyecto Hoshimi (PH) es un juego que apunta a promover el uso creativo de los lenguajes de programación y otras herramientas. PH es útil para: i) Acercar a los jóvenes a las tecnologías de la información mediante un juego ameno; ii) Conceptualizar la programación como una tarea creativa a través de una simulación estratégica y iii) Enseñar tanto conceptos básicos de programación y lógica, como el paradigma de orientación a objetos con tecnología .Net.

En términos generales, PH es un juego cuyo objetivo principal es curar enfermedades del cuerpo humano, aunque también existen otros objetivos secundarios como trasladarse a diferentes puntos o mantener con vida determinados NanoBots. Estos objetivos son llevados a cabo con diferentes NanoBots, que reciben sus órdenes de un NanoBot especial denominado *NanoAI* (Inteligencia Artificial). El comportamiento de cada NanoBot es descripto por una estrategia desarrollada por el alumno con tecnología .Net. El lector interesado en conocer más sobre el juego puede estudiar el trabajo realizado por Luzza et al. [16] ó analizar el material disponible en [15].

5 PH-Asistido

Actualmente, el Proyecto Hoshimi tiene dos formas de desarrollar estrategias:

1. **Usando el editor visual integrado:** este es un editor dedicado que presenta el problema de proveer una escasa expresividad, por ejemplo: i) No posee acciones para declarar y usar variables, ii) Si bien permite el uso de una sentencia condicional, tiene pocas opciones a la hora de elaborar los predicados y iii) El editor no es extensible.

2. **Usando un editor y un compilador de .Net:** esta forma permite sobrepasar las limitaciones indicadas en el ítem anterior, pero exige que el estudiante domine los conceptos de programación orientada a objetos. Además, el estudiante debe lidiar con los errores comunes vinculados con los lenguajes de programación, como los que se explicaron en las secciones previas, que entorpecen el aprendizaje.

Es así que surge PH-Asistido, un editor visual que tiene la sencillez del primer modo y posee el potencial y la flexibilidad del segundo modo. El lenguaje de aplicación del editor es el lenguaje Hoshimi, un LED cuyo dominio es el usado en el juego. En PH-Asistido se mantienen todos los comandos básicos que ya existían en el lenguaje Hoshimi. No obstante, algunos de ellos fueron mejorados y/o extendidos. En primer lugar, se flexibilizaron los comandos relativos al control del flujo. Como por ejemplo, la iteración ahora permite como condición cualquier expresión lógica (antes sólo permitía un entero que correspondía a la cantidad de iteraciones). La instrucción de decisión también fue mejorada (comando similar a un if-then-else) siguiendo un esquema similar al de las iteraciones. Esta característica incrementa el nivel de expresividad de dicha sentencia debido a que se pueden incorporar expresiones más complejas. Además, se agregaron nuevas funcionalidades, las más importantes están relacionadas al trabajo con variables. En PH-Asistido es posible definir variables, cambiarles el valor usando una acción de asignación y utilizarlas como parámetros en los demás comandos. Los tipos disponibles son aquellos involucrados con el dominio, como "Punto" (par <x, y> que simboliza una ubicación) y "Entero" (comúnmente usado para cuantificar) entre otros. Es importante destacar que la acción de asignación posee un control visual que asiste en el armado de la operación. Por un lado, se especifica el lado izquierdo de la asignación, eligiendo primero el tipo de la variable desde una lista y luego el nombre desde otra lista filtrada por el tipo seleccionado. Esto tiene como consecuencia evitar errores sintácticos tanto para el tipo como para el nombre. Por otra parte, se especifica el lado derecho de la asignación con una herramienta que permite el armado de expresiones. Esta herramienta tiene tres pestañas: i) *Variables*, que permite seleccionar de un listado de variables filtradas por el tipo correcto que espera la expresión; ii) *Funciones*, que permite seleccionar de un listado de funciones filtrado por aquellas cuyo tipo devuelto sea el esperado por la expresión; y iii) *Constantes*, que permite especificar una constante literal. En el caso de que se seleccione una función, se repite el proceso para armar las expresiones correspondientes a los parámetros. En otro caso se puede finalizar el proceso. Es importante destacar que lo anteriormente descripto tiene como consecuencia evitar errores sintácticos pero también otros errores semánticos como errores de tipos, etc.

La Construcción Asistida de Expresiones (CAE) mencionada en el párrafo precedente, es reutilizada en todos los procesos de parametrización de todos los comandos utilizados en la estrategia. De esta forma, se garantiza que las expresiones sean válidas. Si bien las extensiones descriptas previamente son suficientes para mejorar los desarrollos, es posible que el usuario necesite algunas más, que están en la API del juego pero no en el editor. Por esta razón, se incluyó la posibilidad de extender las funciones y las variables/propiedades que se pueden utilizar en la CAE. Basta con editar el archivo xml correspondiente (Funciones.xml y Variables.xml) y agregar los elementos necesarios. En el trabajo de Luzza et al. [16] se puede ver tanto cómo extender la herramienta como ejemplos de extensión.

5.1 Estrategias

En PH-Asistido, una estrategia es representada por un conjunto de pestañas. Cada pestaña representa un NanoBot y tiene un panel de acciones que permite especificar el comportamiento de dicho NanoBot. El comportamiento es representado por un árbol de acciones donde la raíz es el mismo NanoBot. Cada acción puede ser parametrizada según corresponda. La acción inicial permite parametrizar el nombre del NanoBot y otras características del mismo. Además permite declarar variables para ser usadas luego en las parametrizaciones de otras acciones.

5.2 Compilación

Una vez creada la estrategia en el editor, sólo resta compilar para obtener el ensamblado final que será usado por el juego. El proceso de compilación recorre dos veces todos los nodos raíz de la estrategia. La primera vez buscando errores. Si hay errores se muestran y el proceso termina. Si no, nuevamente se recorren los árboles invocando en ellos el método "GetCodigo". En ambos casos, se invoca recursivamente sobre los nodos hijos y así sucesivamente hasta llegar a los nodos hoja. Este proceso finaliza con todo el código recolectado libre de errores, el cual se usa con dos fines: 1) Compilarlo en un ensamblado dll y 2) Guardarlo como código fuente para facilitar el pasaje del lenguaje Hoshimi a C#.

6 Caso de Estudio

Un ejercicio habitual en el juego es calcular la entidad más cercana a un NanoBot. Esto es similar a ejercicios básicos de programación donde se busca el número menor o mayor de un arreglo. En este dominio, el cálculo se hace sobre la distancia entre una entidad y un NanoBot. También, a diferencia de un arreglo, se utiliza una lista dinámica (muy utilizadas en el juego) que es más sencilla de manejar puesto que la implementación está encapsulada. El procedimiento consiste en los siguientes pasos:

1. Establecer una distancia mínima de referencia en 400, puesto que el mapa tiene 200 x 200 puntos
2. Recorrer la lista de puntos deseada
    a. Calcular la distancia entre el punto y el NanoBot
        i. Si la distancia es menor a la mínima entonces: i) Actualizar la distancia mínima y ii) Guardar el punto
3. Informar el resultado

El algoritmo puede ser resuelto empleando el NanoBot AI que ya viene por defecto al crear una nueva estrategia con PH-Asistido (Archivo → Nuevo). Todos los comandos utilizados se encuentran en la barra de herramientas a la izquierda del panel de acciones. La traducción del algoritmo anterior expresado en pseudo-código al lenguaje Hoshimi se puede hacer con los siguientes pasos:

1. En la acción raíz, usar el área de parametrización (panel derecho) para agregar dos variables enteras (“dist” para la distancia mínima encontrada y “aux” para los cálculos auxiliares de distancia) y dos variables tipo punto (“ptoCercano” para el punto más cercano encontrado y “pto” para trabajo auxiliar).
2. Agregar un estado, que será usado para calcular el punto más cercano.
   a. Establecer el valor mínimo de referencia para la variable “dist” en 400 con una acción de asignación.
   b. Recorrer la lista de puntos deseada (por ejemplo “ListaHP”, que es provista por el sistema) con la variable “pto”. Ambas especificaciones se establecen en la parametrización de la acción “Recorrer” que es similar a una iteración “foreach” la cual recorre una colección dando a una variable auxiliar cada ítem de la misma.
      i. Asignar en “aux” la distancia entre “pto” y la ubicación del NanoBot (“Ubicación_AI” en el caso de hacerlo con la AI). Esta operación se especifica en la parametrización de la asignación usando la CAE bajo la pestaña “Funciones” con la función “Distancia” que pide dos puntos como parámetros.
      ii. Preguntar si la distancia recién calculada (aux) es menor que la de referencia (dist). Esto se realiza con una acción de decisión ? y especificando en la parametrización la función “Menor” entre “aux” y “dist”
      iii. En caso de ser verdadera la proposición anterior, actualizar la distancia de referencia y el punto más cercano, con sendas acciones de asignación.
   c. Una vez recorrida la lista, cambiar al siguiente estado (Estado 2) para informar los resultados. Esto se realiza con una acción de “Cambiar de Estado” y especificando el estado al que se desea cambiar.
3. Agregar otro estado (Estado 2) para reportar los resultados
   a. Escribir un cartel para anunciar el resultado. Para escribir se utiliza la acción “Escribir” y se especifica lo que se desea escribir en la parametrización usando la CAE. En este caso se uso la pestaña “Constante” para escribir “El punto más cercano es:”.
   b. Escribir el contenido de la variable “ptoCercano”. Una vez más utilizando la acción “Escribir” pero esta vez se utiliza la función “Pto_Txt” que convierte un punto pasado como parámetro a texto.

A continuación (figura 1.a) se muestra el programa desarrollado en el lenguaje Hoshimi con PH-Asistido para resolver el problema.

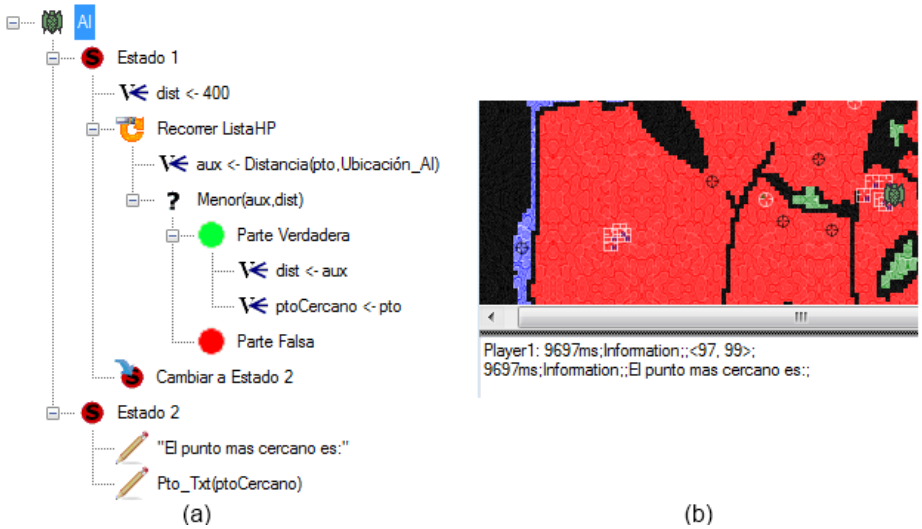

Fig. 1. (a) Programa en lenguaje Hoshimi y (b) su correspondiente resultado

```
public partial class MiEquipo : Player, IAccion{
 public void Accion(){
  switch (___ESTADO){
   case "Estado 1":
         dist = 400;
         foreach (Punto ___Item in ListaHP){
                 pto = ___Item;
                 aux = Distancia(pto, Ubicación_AI);
                 if (Menor(aux, dist)){
                         dist = aux;
                         ptoCercano = pto;
                 }
                 else{}
         }
         ___ESTADO = "Estado 2";
         break;
  case "Estado 2":
         MiUtils.Escribir("El punto mas cercano es:");
         MiUtils.Escribir(Pto_Txt(ptoCercano));
         break;
  }//fin switch
 }//fin acción
 string ___ESTADO = "Estado 1";
 public Entero aux = 0, dist = 0;
 public Lista<Punto> miLista = new Lista<Punto>();
public Punto pto = default(Punto), ptoCercano = default(Punto);}
```

Fig. 2. Código C# generado por PH-Asistido

El proceso de compilación genera: i) El código C# correspondiente al algoritmo (ver figura 2), y ii) Un archivo ensamblado dll que puede ser utilizado en el juego

para “ejecutar” la estrategia y verificar la solución. En la figura 1.b se puede apreciar el resultado de ejecutar la estrategia en el juego.

7 Conclusión

En este artículo se presentó PH-Asistido, una herramienta que combina un editor dirigido por la sintaxis con un lenguaje específico del dominio: el Lenguaje del Proyecto Hoshimi, un juego útil para la enseñanza de la programación. PH-Asistido intenta paliar dos grandes dificultades, que se presentan en el momento de enseñar programación a estudiantes, como lo son: i) Encontrar un tema entretenido y desafiante, y ii) Lidiar con la sintaxis y el idioma de los lenguajes de programación tradicionales. Para alcanzar este objetivo, PH-Asistido implementa un editor visual que asiste al usuario a construir las sentencias libres de errores de sintaxis. Además, utiliza y mejora el Lenguaje del Proyecto Hoshimi. Las mejoras realizadas al lenguaje se relacionan con la introducción de nuevos comandos que posibilitan trabajar con variables, selecciones, iteraciones, etc., facilitando de esta forma la tarea de programar. Las características mencionadas previamente permiten que el alumno pueda centrarse en aprender conceptos de lógica y programación, sin la distracción causada por corregir errores comunes como falta de símbolos de puntuación o la utilización de palabras claves que no están en su idioma natural. PH-Asistido también crea un puente entre el juego y el aprendizaje de POO, ya que para cada programa escrito en el lenguaje del Proyecto Hoshimi, se genera el código C# equivalente. Esta peculiaridad permite que el alumno avanzado pueda relacionar: el código del programa escrito en el lenguaje del juego y el código orientado a objeto que verdaderamente se ejecuta. De esta forma el estudiante puede ir internalizando las construcciones básicas provistas por los lenguajes orientados a objetos.

El trabajo futuro está orientado en dos direcciones principales. La primera consiste en la elaboración de diferentes tests que permitan medir las mejoras que se producen en la enseñanza de la programación cuando se utiliza PH-Asistido para dicha tarea. La segunda, se centra en aplicar la estrategia usada en la creación de PH-Asistido, en nuevos editores similares para los lenguajes de programación vistos en las materias introductorias de programación. Estos lenguajes muchas veces son vistos como pseudo-código y por ende carecen de un ámbito de ejecución para verificar los resultados.

8 Referencias

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

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

[3]. D. Wile. Lessons learned from real DSL experiments. Science of Computer Programming, Vol 51, Issue 3, pages 265-290, 2004.

[4]. Krzysztof Czarnecki and Ulrich W. Eisenecker, Generative Programming, Addison-Wesley, 2000, ISBN 0-201-30977-7.
[5]. Seymour Papert. Mindstorms: children, computers, and powerful ideas. Basic Books, Inc., New York, NY, USA, 1980.
[6]. Daniel Watt. Learning With Logo. McGraw Hill, 1983.
[7]. MollyWatt and DanielWatt. Teaching with Logo: Building Blocks for Learning. Addison-Wesley Pub, 1986.
[8]. J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. The scratch programming language and environment. ACM Transactions on Computing Education, 10(4):1–15, 2010.
[9]. M. Resnick, Y. Kafai, and J. Maeda. A networked, media-rich programming environment to enhance technological fluency at after-school centers in economically-disadvantaged communities. MIT Media Laboratory, Proposal to National Science Foundation (Information Technology Research), 2003.
[10]. Raul Medina-Mora. Syntax-Directed Editing: Towards Integrated Programming Environments. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1982.
[11]. Javier Gonzalez Sanchez, Ramiro A. Berrelleza Perez, and Maria Elena Chavez Echeagaray. Introducing computer science with project hoshimi. In Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion, OOPSLA ’07, pages 908–914, New York, NY, USA, 2007. ACM.
[12]. M. Mernik, J. Heering, T. Sloane. When and How to Develop Domain-Specific Languages. ACM Computing Surveys, 37(4es), December 2005.
[13]. 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.
[14]. S. Georgantaki, S. Retalis. Using educational tools for teaching object oriented design and programming. Journal of Information Technology Impact (Jiti), 7(2):111–130, 2007.
[15]. Gaming.Net. Disponible online en http://www.pilp.edu.ar/PILPAsp/Paginas/Pagina.asp?PaginaPilpId=22. Visto en mayo de 2012.
[16]. M. Luzza, M. Berón, P. Henriques. PH-Helper - a Syntax-Directed Editor for Hoshimi Programming Language, HL. SLATe'12, pages 71-89 Symposium on Languages, Applications and Technologies, 2012.
[17]. S. Goldrei. The Design, Implementation and Use of Domain Specific Languages. 2007.
[18]. J.C. Cleaveland. Building application generators. IEEE Software, 1988.
[19]. T. Teitelbaum. The cornell program synthesizer: a syntax-directed programming environment. SIGPLAN Not., 14(10):75–75, Oct. 1979