Método y Entorno Integrado de Desarrollo para el Aprendizaje en Lógica de Programación Orientada por Objetos

Carlos Arturo Castro Castro
Facultad de Ingeniería, Universidad de San Buenaventura Medellín
Medellín, Antioquia 050010, Colombia

Gabriel Enrique Taborda Blandón
Facultad de Informática, Tecnológico de Antioquia Institución Universitaria
Medellín, Antioquia 050034, Colombia

Ricardo de Jesús Botero Tabares
Facultad de Informática, Tecnológico de Antioquia Institución Universitaria
Medellín, Antioquia 050034, Colombia

RESUMEN

El artículo presenta una didáctica para apoyar el proceso de enseñanza-aprendizaje de la lógica de programación con orientación a objetos, sustentada en el proyecto "Método Integrado de Programación Secuencial y Orientada a Objetos – MIPSOO" y en un entorno integrado de desarrollo implementado con tecnología Java denominado "Sistema para el Modelamiento por Objetos –SISMOO", que permite editar, compilar y ejecutar programas escritos en el seudo lenguaje definido en MIPSOO.

El Método Integrado de Programación Secuencial y Orientada a Objetos, reúne elementos pedagógicos del aprendizaje basado en problemas y de la programación orientada a objetos para propiciar un ambiente educativo que disminuya las barreras generadas entre los aprendizajes de la lógica tradicional secuencial y de la programación orientada a objetos.

Los proyectos contaron con la participación de investigadores del Tecnológico de Antioquia, Universidad de San Buenaventura Medellín, Fundación Universitaria Católica del Norte y Fundación Universitaria Luis Amigó – Medellín, y se encuentra en proceso de implementación y verificación de resultados de aprendizaje.

Palabras Clave—Programación de computadores, Algoritmos, Ingeniería de Software, Aprendizaje, Entorno Integrado de Desarrollo, Lógica de Programación Orientada por Objetos, Seudo Lenguaje, Traductor, Compilador.

I. INTRODUCCIÓN

El aprendizaje de la lógica de programación orientada a objetos adquiere relevancia en tanto que los estándares internacionales así como las herramientas para el desarrollo de software con altos índices de calidad y desempeño, se basan en este paradigma. Adicionalmente se ha evidenciado que el aprendizaje de la lógica de programación con el método tradicional –secuencial (también llamado estructurado) genera en los estudiantes de primer año de Tecnología e Ingeniería de Sistemas y/o Informática, problemas de comprensión y motivación en los cursos posteriores de programación orientada a objetos. Investigadores de varias universidades de Antioquia-Colombia, desarrollaron el **"**Método Integrado de Programación Secuencial y Orientada a Objetos – MIPSOO**"** que integra elementos didácticos, sintácticos y pedagógicos para favorecer procesos de enseñanza-aprendizaje de la lógica y programación orientada a objetos, liderado por la institución universitaria Tecnológico de Antioquia. MIPSOO propone un seudo lenguaje orientado a objetos con características similares a lenguajes de producción como Java, C#, C++ y Visual Basic.Net; contiene además elementos pedagógicos del aprendizajes significativo y basado en problemas. Como complemento a MIPSOO, se desarrolló un segundo proyecto de construcción de una herramienta de software tipo entorno integrado de desarrollo (IDE por sus siglas en inglés), denominado **"**Sistema para el Modelamiento por Objetos **–**SISMOO**",** que procesa programas escritos en seudo lenguaje y los traduce al lenguaje Java. Se pretende que este software traduzca también al lenguaje C# o a Visual Basic.Net.

Ambos proyectos han generado libro de investigación [1], reforma curricular en las áreas de algoritmos y estructura de datos [2], artículos en revistas [3] y ponencias internacionales [4].

Este artículo está organizado de la siguiente manera:

- Un marco teórico donde se exponen de manera general los principales conceptos de la orientación a objetos y herramientas de aprendizaje basadas en software para dicho paradigma, así como conceptos sobre compiladores.
- El seudo lenguaje propuesto en el MIPSOO.
- El IDE SISMOO.
- La socialización de los proyectos en eventos nacionales e internacionales.
- Conclusiones y referencias.

II. MARCO TEÓRICO

En términos del paradigma de programación orientado a objetos, un objeto es un elemento o instancia de una clase. Una clase se define como "una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica" [5]. Los atributos pueden ser de tipo primitivo o de un tipo abstracto de dato. Las operaciones definidas sobre un objeto determinan su comportamiento. Una relación describe un conjunto de enlaces, los cuales son conexiones entre objetos. La semántica de una clase determina lo que las instancias de esa clase hacen y cómo lo hacen; dicha semántica se puede definir de manera informal mediante responsabilidades o de manera formal

mediante el Lenguaje de Restricciones de objetos (OCL).

Los objetos tienen una identidad y un estado dado por los valores actuales de sus atributos. Un objeto se comunica con otro por medio de mensajes, que obligan la ejecución de una operación por el objeto que lo recibe. Las operaciones sobre los objetos se pueden sobrecargar, facilitan el polimorfismo y constituyen la interfaz de comunicación.

Los lenguajes de programación clásicos como C, Pascal, Fortran, Cobol y Basic, que en sus comienzos soportaban el paradigma imperativo, han trascendido al paradigma orientado a objetos y se han caracterizado por su hegemonía en el mercado del software.

Algunos métodos, estudios y reflexiones para el aprendizaje de la programación han sido desarrollados por los profesores Gayo [6], Sánchez [7], Zapata [8] y Taborda [9]; también se han ejecutado proyectos como SHABOO de la Universidad Industrial de Santander [10] y Cupi2 de la Universidad de Los Andes [11].

Los primeros lenguajes de programación vinieron acompañados del término *compilador*, que en términos generales se puede definir como "Un programa que lee otro programa escrito en un lenguaje fuente, y lo traduce a un programa equivalente en otro lenguaje, lenguaje objeto (figura 1). Como parte importante de este proceso de traducción, el compilador informa a su usuario de la presencia de errores en el programa fuente." [12]. Otra forma de visualizar un compilador es a través de su representación simbólica (figura 2), donde en cada extremo se identifican los diferentes lenguajes de programación que intervienen: la letra A corresponde al lenguaje fuente o inicial, C es el lenguaje objeto o final y B es el lenguaje en el que está construido el compilador [12].

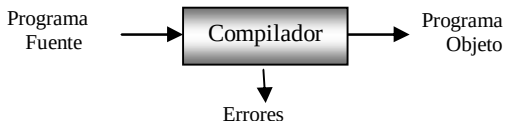

Fig. 1. Representación esquemática de un compilador

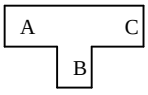

Fig. 2. Representación simbólica de un compilador

En la construcción de un compilador intervienen varias áreas de conocimiento como la arquitectura de computadores, la teoría de lenguajes, las maquinas de estado finito, los algoritmos y la ingeniería de software. Para comprender mejor su funcionamiento, diseño y elaboración se suele dividir en seis fases: Análisis de Léxico o scanner, Análisis gramatical y sintáctico o parser, Análisis semántico y de verificación de tipos, Generación de código intermedio, Optimización de código intermedio, y Generación de código objeto o final. A Las tres primeras se les conoce con el nombre de etapa de análisis y a las tres últimas como etapa de síntesis [13]

III. EL SEUDO LENGUAJE DE MIPSOO

Los elementos pedagógicos involucrados en MIPSOO tienen en cuenta la teoría del *aprendizaje significativo* de Ausubel, en la que un nuevo material adquiere significado para el sujeto a partir de su relación con conocimientos anteriores, es decir el aprendizaje es significativo cuando puede incorporarse a las estructuras de conocimiento que posee el sujeto [14] y del modelo de *aprendizaje basado en problemas* (ABP) que se puede definir como un proceso de indagación, curiosidades, dudas e incertidumbres sobre fenómenos complejos de la vida [15]. La didáctica de MIPSOO plantea un conjunto de problemas clasificados en cuatro tipos por orden de volumen y complejidad:

i)Problemas cuya solución implica el uso de una clase, denominada *Proyecto*: en estos casos se trabaja con clases vacías, es decir, clases que carecen de atributos, con uno o varios métodos, uno de ellos denominado *principal* de carácter obligatorio, donde es innecesaria la creación de objetos durante el proceso de solución. Lo que se persigue con este tipo de problemas es aprender a manejar sentencias de control y subprogramas, conceptos propios de la programación imperativa. Además, esto apoya el hecho que el método de aprendizaje es mixto, es decir, se fundamenta en la programación orientada a objetos pero apoya a la programación procedimental como un escaño previo a la profundización y modelado por objetos.

ii) Problemas cuya solución implica el uso de dos clases, una del ámbito del problema y otra del ámbito de la solución (*Proyecto*): en este tipo de problemas se aplican los conceptos de atributos encapsulados, métodos para establecer y obtener el estado de los objetos, métodos constructores, creación de objetos y paso de mensajes.

iii) Problemas con varias clases relacionadas entre sí, empaquetadas y ejemplarizadas desde la clase *Proyecto*: se profundiza en este caso el concepto de asociación entre clases, originado porque un atributo es un objeto de una clase, y no de tipo primitivo.

iv) Problemas cuya solución involucra mecanismos de herencia: en estos problemas se tratan los conceptos de clase abstracta, clase base, clase derivada, clase final, interfaces y polimorfismo.

La solución de cada problema conlleva varias etapas: construcción de la tabla de requerimientos, abstracción de clases (diagramas conceptual o de clases según formalismo del Lenguaje de Modelado Unificado – UML), descripción de los contratos de clase y escritura de seudo código orientado a objetos, tal como lo ilustra el siguiente ejemplo:

> "La famosa ecuación de Einstein para conversión de una masa m en energía viene dada por la fórmula $E = mc^2$, donde c es la velocidad de la luz, $c = 2.997925 \times 10^{10}$ m/seg. Leer una masa en gramos y obtener la cantidad de energía producida cuando la masa se convierte en energía.
> Observación: si la masa se da en gramos, la fórmula produce la energía en ergios."

El problema conlleva los requerimientos R1 y R2 descritos en la tabla 1, un diagrama de clases expuesto en la figura 3, una descripción de los contratos de clase en las tablas 2 y 3 y el seudo código expuesto en la figura 4.

TABLA 1. REQUERIMIENTOS FUNCIONALES

Req.	Descripción	Entradas	Salidas
R1	Ingresar información sobre la masa del objeto	La masa del objeto (dada en gramos)	Un nuevo objeto con masa igual a la ingresada por el usuario
R2	Obtener la cantidad de energía producida por un objeto.	La masa del objeto (en gramos).	La energía del objeto (en ergios).

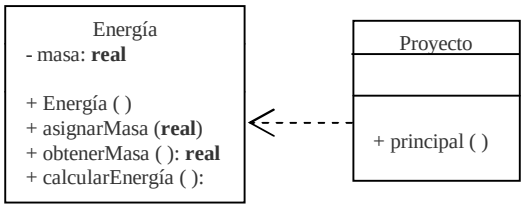

Fig. 3. Diagrama de clases

TABLA 2. CONTRATO DE LA CLASE ENERGÍA

Método	Requisito asociado	Precon_ dición	Poscondición
principal ( )	R1, R2	No existe objeto.	Se conoce un objeto con determinada masa en gramos y la energía que disipa.

TABLA 3. CONTRATO DE LA CLASE PROYECTO

Método	Requisito asociado	Precondición	Poscondición
Energía ( )	R1	No existe un objeto para calcular su energía.	Existe un objeto listo para calcular su energía, con una masa inicial igual a 0 (cero).
asignarMasa (**real**)	R2	El objeto tiene un valor establecido para el atributo *masa*, el cual será modificado con el valor del parámetro.	El atributo *masa* tiene un nuevo valor, es decir, se ha modificado el estado del objeto.
obtenerMasa ( ): **real**	R2	El objeto de tipo Energía tiene una masa definida y desconocida al instante.	Se retorna el valor actual de la masa del objeto (se conoce su estado).
Calcular_ Energía ( ): **real**	R2	El objeto receptor del mensaje, de tipo Energía, tiene definida su masa.	Se conoce la energía desplegada por el objeto.

La estructura general de una solución objetual a un problema encaja con el patrón de la figura 5:

```
//Enunciado del problema (a modo de comentario, opcional)
[Sentencias importar ]
[Definición de clases del ámbito de la solución]
clase Proyecto
      estático principal( )
            // Cuerpo del método principal
      fin_método
fin_clase
```

Fig. 5. Estructura general de una solución orientada a objetos

```
clase Energía
      privado:
            real masa
      público:
            asignarMasa (real m)
            masa = m
       fin_método
       //-------------------------------
       real obtenerMasa ( )
            retornar masa
       fin_método
       //-------------------------------
       real calcularEnergía ( )
            real e
            constante real c = 2.997925 * Mat.elevar (10, 10)
            e = masa * Mat.elevar (c, 2)
            retornar e
       fin_método
fin_clase
//*********************************************
clase Proyecto
      estático principal ( )
            Energía e = nuevo Energía ( )
            Flujo.imprimir ("Ingrese una masa en gramos:")
            real ms = Flujo.LeerReal ( )
            e.asignarMasa(ms)
            Flujo.imprimir ("La energía del objeto con masa " +
                  e.obtenerMasa( ) + " gramos es " +
                  e.calcularEnergía ( ) + " ergios")
      fin_método
fin_clase
```

Fig. 4. Seudo código

Para el introducir el concepto de reutilización se sugiere al docente plantear un esquema básico de clases de uso común que se puede maneja desde un comienzo a modo de marco de trabajo, posibilitando el uso de clases predefinidas como se observa en el ejemplo anterior donde se invoca al método estático *elevar* de la clase *Mat*. Este marco de trabajo se puede aumentar con otras clases importantes para la solución de un determinado grupo de problemas. En la figura 6 se muestra el paquete con clases de uso común.

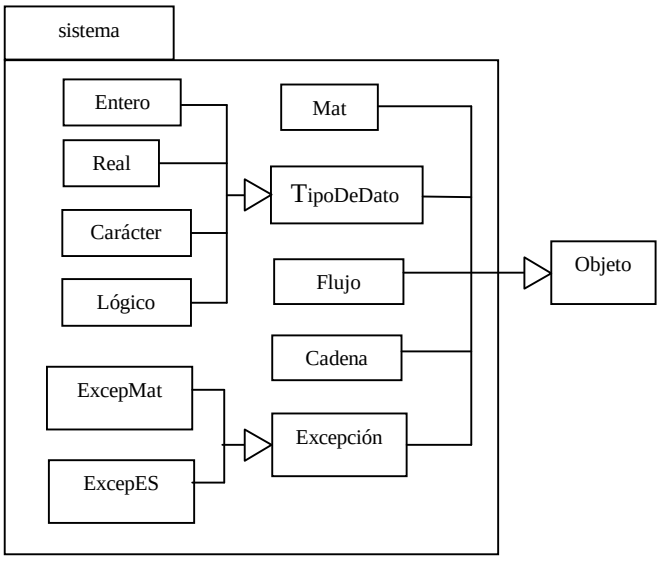

Fig. 6. Paquete de clases de uso común

Los elementos sintácticos del seudo lenguaje orientado a objetos incluyen un listado de 58 palabras reservadas; cinco tipos de datos (entero, real, lógico, carácter y cadena); operadores aritméticos, lógicos, relacionales y de asignación (se retoman los operadores del lenguaje C); las sentencias de control convencionales; y elementos estructurales para la definición de una clase, un método, un paquete y una interfaz, cuya estructura general se presenta en las figuras 7, 8, 9 y 10, respectivamente.

```
[abstracto | final] [público | privado | protegido] clase nomClase
            [heredaDe nomClaseBase] [implementa nomInterfaz]
        // Cuerpo de la clase
fin_clase
```

Fig. 7. Estructura de una clase

```
[estático][<tipo_devuelto>] nomMétodo([argumentos] )
    [ público:
        // Miembros públicos]
    [ privado:
        // Miembros privados]
fin_método
```

Fig 8. Estructura de un método

```
paquete nomPaquete
        // Cuerpo del paquete
fin_paquete
```

Fig. 9. Estructura de un paquete

```
interfaz nomInterfaz
        // Cuerpo de la interfaz
fin_interfaz
```

Fig.10. Estructura de una interfaz

IV. EL IDE SISMOO

El entorno integrado de desarrollo SISMOO tiene la apariencia de la figura 11, donde se observa una ventana con un menú principal, un barra de iconos o herramientas, un panel lateral izquierdo para exploración de archivos, un panel lateral derecho que hace las veces de editor que puede desplegar varios archivos a la vez y un panel inferior para la emisión de mensajes al usuario como los de compilación.

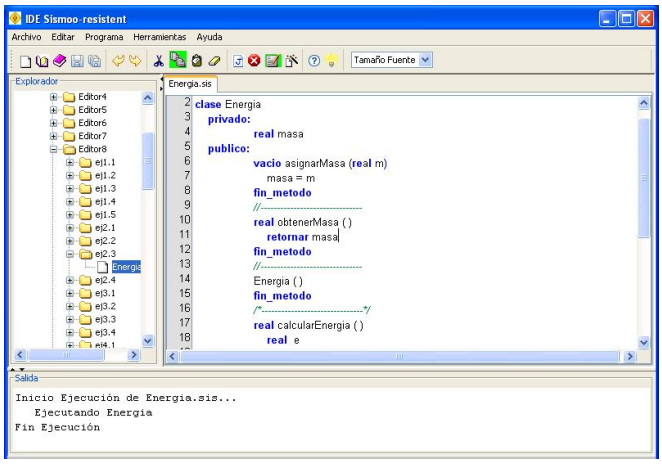

Fig. 11. Ambiente de trabajo SISMOO

Los resultados que emiten los programas se presentan en ventanas independientes a la ventana principal (figura 12). Además, se cuenta con la opción de traducción a lenguaje Java y se proyecta la traducción a otro lenguaje como Visual Basic.Net o C#.

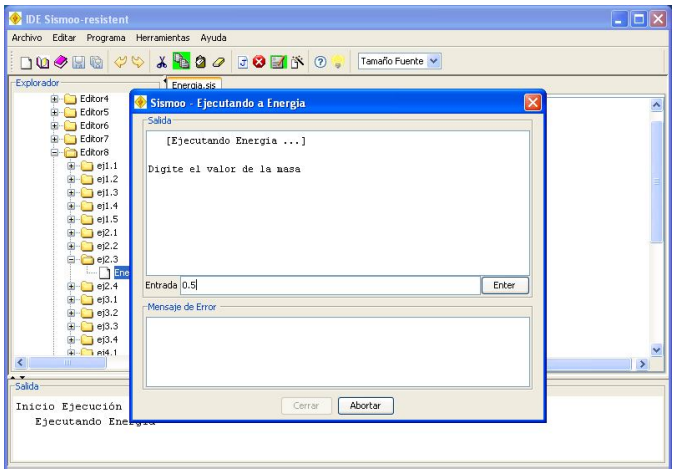

Fig. 12. Entorno de ejecución de SISMOO

En la figura 13 se observa la representación simbólica de SISMOO, los extremos ilustran los lenguajes que intervienen en el proceso, como se explico en la parte II.

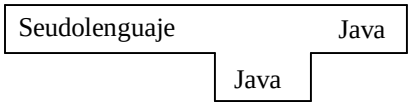

Fig. 13. Representación simbólica de SISMOO

La representación esquemática de SISMOO y su relación con el lenguaje de programación Java (figura 14), implica que su funcionamiento requiere de la instalación previa del Kit de Desarrollo de Java (JDK), para las labores de compilación y traducción a Bytecode (que la realiza la aplicación Javac) y ejecución en la maquina virtual de Java (JVM).

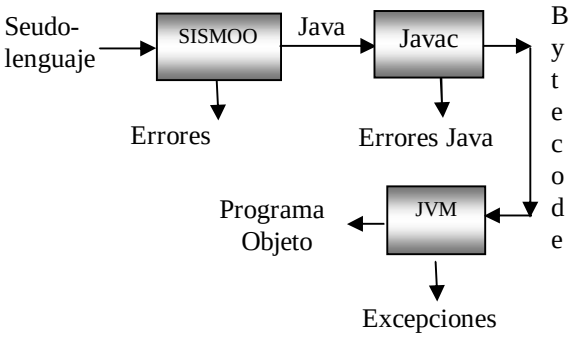

Fig. 14. Representación esquemática de SISMOO y su relación con Java

SISMOO no es solamante un traductor a lenguaje Java, sino un entorno integrado de desarrollo que permite al usuario realizar tareas de edición (con los comandos de copiar, pegar, cortar, deshacer, rehacer y seleccionar todo) y manejo de archivos (abrir, guardar, cerrar y nuevo). También cuenta con el

manejo de auto formatos en forma dinámica, que consiste en resaltar con formatos especiales las palabras claves, las cadenas de caracteres o literales y los comentarios, utilidad que es de gran ayuda para los desarrolladores porque les permite corregir los errores de digitación con mayor agilidad. La funcionalidad mas importante del IDE consiste en traducir, compilar y ejecutar programas escritos en seudo lenguaje.

Entre los posibles lenguajes para realizar la traducción, se selecciono Java por varias razones: es un lenguaje puro orientado a objetos, potente, de arquitectura neutra, de libre uso y ampliamente utilizado en aplicaciones corporativas y orientadas a la Web.

El modelo de desarrollo de Ingeniería del Software que se empleo para la construcción de SISMOO es cercano al enfoque evolutivo por prototipos [16]. El primer prototipo que se diseño fue un editor de texto sencillo, al segundo prototipo se le adicionaron las funcionalidades de compilar y ejecutar programas escritos en lenguaje Java transformándolo en un rudimentario IDE para Java, a continuación se abordo la tarea de traducción del Seudo lenguaje y así sucesivamente hasta llegar prototipo actual, el nueve, con las características descritas en este artículo, que se encuentra en periodo de pruebas y ajustes para el lanzamiento a principios de 2010 de la versión 1.0 de SISMOO como software abierto de distribución gratuita.

Algunos aspectos del diseño de SISMOO se presentan en los diagramas de clase de las figuras 15 y 16, descritos en la tabla 4.

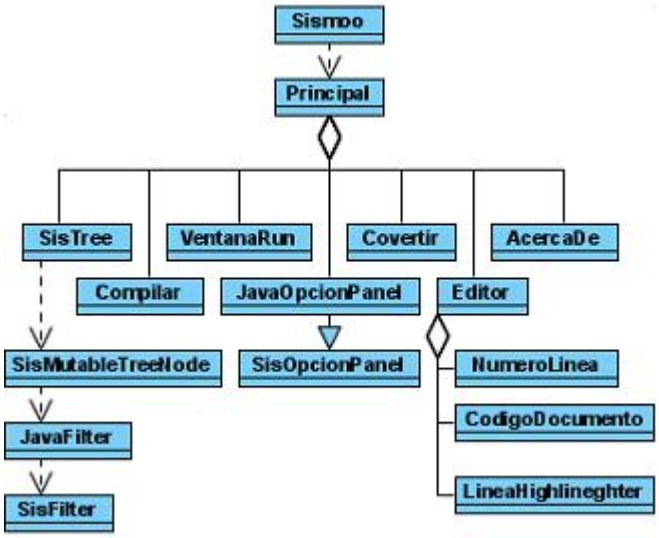

Fig. 15. Diagrama de clases para SISMOO

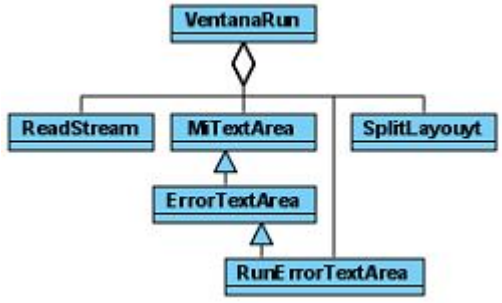

Fig. 16. Extensión del diagrama de clases para SISMOO

TABLA 4. DESCRIPCIÓN DE CLASES PARA SISMOO

Nombre	Deriva de	Breve Descripción
SISMOO	JFrame	Ventana de inicio
Principal	JFrame	Es la clase central que contiene el ambiente de trabajo de SISMOO
SisTree	JTree	Implementa el explorador
SisMutable TreeNode	DefaultMutable TreeNode	Para manejar los nodos o carpetas en el explorador
JavaFilter	FileFilter	Filtro general para los archivos
SisFilter	FileFilter	Filtra los archivos tipo SISMOO, con extensión punto sis (.sis)
Editor	JPanel	Panel principal para el editor de texto
SismoText Panel	JPanel	Panel para las carpetas de cada archivo
Numero Linea	JComponent	Maneja la numeración de las líneas de código
Codigo Documento	DefaultStyled Document	Reconoce los elementos que poseen formato especial
lineaHighlight er	DefaultHighlighter	Para resaltar el texto seleccionado
MenuPopup	JPopupMenu	Menú contextual del área de edición
VentanaRun	JDialog	Despliega la venta de ejecución
MiJTextArea	JScrollPane	Maneja el área de texto de salida en la ventana de ejecución
ErrorTextArea	MiJTextArea	Captura las excepciones o errores en tiempo de ejecución
RunErrorText Area	ErrorTextArea	Maneja el área de texto de errores en tiempo de ejecución
ReadStream	Runnable	Controla la caja de texto de lectura en tiempo de ejecución
SplitLayout	LayoutManager	Para el diseño de los panel en la ventana de ejecución
SisActivo Panel	JPanel	Panel para la ventana de configuración
JavaOpcion Panel	SisActivoPanel	Ventana de configuración de Kit de Desarrollo de Java
Convertir	Object	Realiza la traducción a Java
Compilar	Thread	Ejecuta la compilación en Java
AcercaDe	JFrame	Ventana de Ayudas

V. SOCIALIZACIÓN

Los proyectos MIPSOO y SISMOO se han socializado en varias actividades:

- Cursos de extensión dirigidos a profesores del área informática y a profesores de la media técnica.
- Diseño curricular del Módulo Desarrollo del Pensamiento Analítico Sistémico I, dentro del proyecto Alianza Futuro Digital Medellín [17] donde intervienen instituciones de educación superior y representantes del sector productivo, con el liderazgo de la Secretaría de Educación de Medellín y la interventoría del Ministerio de Educación Nacional.
- Conferencias en instituciones de educación superior, como el Tecnológico de Antioquia, UMECIT en ciudad de Panamá, dirigidas a estudiantes y profesores de dichas instituciones.
- Curso “Enseñanza de la programación orientada a objetos,

una introducción a la propuesta metodológica de GIISTA", en la Universidad FASTA, Mar del Plata, Argentina.

- Ponencia "Método de aprendizaje en fundamentos de programación con orientación a objetos", en el marco de la XXVII Reunión Nacional y el VI Encuentro Iberoamericano "El profesor de ingeniería, profesional de la formación de ingenieros", organizado por ACOFI y ASIBEI [4].
- Artículo en la Revista Virtual de la Universidad Católica del Norte [3].
- Libro producto de investigación [1].

VI. CONCLUSIONES

Si bien los proyectos MIPSOO Y SISMOO no se consideran terminados debido a que se encuentran en etapa experimental y de divulgación, la comunidad académica ha manifestado interés sobre los mismos, ya que existen inconsistencias entre la formación en lógica de programación tradicional –secuencial y los cursos específicos en programación orientada a objetos, requiriendo de didácticas y herramientas que favorezcan la coherencia curricular, el aprendizaje y la motivación.

El aprendizaje y la enseñanza de los fundamentos de programación de computadoras deben basarse en los paradigmas para desarrollo de software imperantes en el mercado. El uso de herramientas como compiladores y traductores con sus respectivos entornos integrados de desarrollo y de objetos virtuales de aprendizaje, proporcionan insumos de motivación para que las nuevas generaciones de estudiantes de tecnología e ingeniería de sistemas y áreas afines incursionen con éxito en el atractivo mundo del desarrollo de software.

REFERENCIAS

[1] R. Botero, C. Castro, G Taborda, E. Parra y F Osorio. "Fundamentos de Programación con Orientación a Objetos", Grupo GIISTA, Dirección de Investigación y Extensión, Tecnológico de Antioquia, , Medellín, 2006

[2] Acta de Comité Curricular Nº 02, marzo 2 de 2007, Facultad de Informática, Tecnológico de Antioquia.

[3] C. Castro, R. Botero, y E. Parra, "Método integrado de programación secuencial y programación orientada a objetos para el análisis, diseño y elaboración de algoritmos", Revista Virtual de la Universidad Católica del Norte, edición Nº 17, 2006.

[4] E. Parra, C. Castro y R. Botero, póster T1-060 "Método de aprendizaje en fundamentos de programación con orientación a objetos", XXVII Reunión Nacional de Facultades de Ingeniería y VI Encuentro Iberoamericano de Instituciones de Enseñanza de la Ingeniería, Cartagena de Indias, 2007.

[5] G. Booch et al, "El Lenguaje Unificado de Modelado", Addison Wesley Iberoamericana, Madrid, 1999.

[6] D. Gayo et al, "Reflexiones y experiencias sobre la enseñanza de POO como único paradigma". (2005, Octubre 4). [En línea] Disponible: http://www.di.uniovi.es/~dani/publications/jenui03.pdf

[7] J. L. Sánchez., "Algoritmos y Estructura de Datos", Universidad de Antioquia, Facultad de Educación. (2006, Junio 18). [En línea] Disponible: http://ayura.udea.edu.co/~jlsanche

[8] J. D. Zapata, "'El cuento' y su papel en la enseñanza de la orientación por objetos", Proyecto CONEXIONES-Universidad EAFIT, 1998. http://www.c5.cl/ieinvestiga/actas/ribie98/146.html

[9] G. E. Taborda, "Lenguajes III y lenguajes IV", Tecnológico de Antioquia. (2007, Abril 18). [En línea] Disponible: http://www.usuarios.lycos.es/gabtab http://www.usuarios.lycos.es/gabtab1

[10] Grupo GUIA de la UIS, "Sistema Hipermedia Adaptativo para la Enseñanza de la Programación Orientada a Objetos SHABOO". http://lsm.dei.uc.pt/ribie/docfiles/txt2003326195840A016.pdf

[11] Cupi2. Buscando nuevas maneras de aprender a programar. Universidad de los Andes, http://cupi2.uniandes.edu.co/inicio.php

[12] A. Aho, R. Sethi y J. Ullman, "Compiladores: principios, técnicas y herramientas". Addison Wesley, México, 1998.

[13] M. Alfonseca, M. De la Cruz, A. Ortega y E. Pulido, "Compiladores e Intérpretes: Teoría y práctica". Pearson, España , 2006.

[14] J. I. Pozo. "Teorías cognitivas de aprendizaje", Madrid, Ediciones Morata, 1989.

[15] J. Azpilicueta y A. Ledesma. "Constructivismo en la Educación de las Ciencias de la Computación. Una propuesta de Enseñanza-Aprendizaje en Aula Virtual Basada en Resolución de Problemas", VIII Congreso de Educación a Distancia CREAD MERCOSUR / SUL 2004. Córdoba, Argentina.

[16] I. Sommerville," Ingeniería del Software", Pearson Educación, Madrid, 2005.

[17] Alianza Futuro Digital Medellín. (2008, Febrero 3). [En línea] Disponible: www.mineducacion.gov.co/cvn/1665/articles-137378_archivo_pdf6.pdf