Alternativa de Comprobación sintáctica de VLP*: $Gsig\_Parsing$. Aspectos formales y el caso de estudio: $E\_GraPiCO$.

Carlos Andrés Tavera Romero
Universidad de San Buenaventura - Cali; Plan de Ingeniería de Sistemas
catavera@usb.edu.co

and

Juan Francisco Díaz Frías
Universidad del Valle; Escuela de Ingeniería de Sistemas
jdiaz@eisc.univalle.edu.co

and

Aybert Soto Vargas
Universidad Cooperativa de Colombia-Cali; Programa de Ingeniería de Sistemas
saybert01@yahoo.es

and

Javier Andrés Gallego Varona
Universidad Cooperativa de Colombia-Cali; Programa de Ingeniería de Sistemas
Javan_jl@yahoo.es

and

Ánderson Jojoa Giraldo
Universidad Cooperativa de Colombia-Cali; Programa de Ingeniería de Sistemas
anderson_jojoa@yahoo.es

Abstract

This publication presents $Gsig\_Parsing$, an alternative syntactic testing mechanism for visual languages through the division of parsing in two stages, context-free and context-dependent analisis. As a real example of the new proposed form, the visual programm editor E_GraPiCO is shown.

Keywords: Visual languages, Computational calculus, Parsing.

Resumen

En esta publicación se presenta $Gsig\_Parsing$, una alternativa de un mecanismo de comprobación sintáctica para lenguajes visuales por medio de la división del parsing en dos etapas, los Análisis Independientes de Contexto y los Sensibles al Contexto. Como un ejemplo real de la nueva forma propuesta se muestra el editor de programas visuales E_GraPiCO.

Palabras clave: Lenguajes Visuales, Cálculo computacional, Análisis sintáctico.

*Lenguajes de Programación Visual

1. Introducción

Cada vez que se requiere la implementación de un procesador de algún lenguaje, siempre se efectúan de alguna manera las etapas de lo que se entiende como compilador: Análisis léxico, Análisis sintáctico, Análisis semántico, Optimización y Generación de código.

Si se tienen n compiladores cruzados[1], y a su vez, escalonados[2] se tendrá una cantidad n de análisis léxicos, sintácticos, ... y, así sucesivamente, como se muestra en la figura 1.

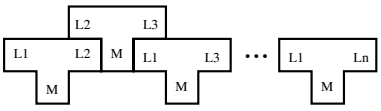

Figura 1: Compilador escalonado.

Aunque las etapas de análisis léxico y sintáctico son bien conocidas y su implementación no presenta un alto nivel de complejidad (por lo menos en las gramáticas de los lenguajes que están dentro de la clase 3 de la jerarquía de Noam Chomsky) prescindir de alguno de estos procesos puede representar una gran ayuda al momento de diseñar y/o implementar un procesador de lenguajes. Para dejar de efectuar el análisis léxico y/o el sintáctico en un compilador B que recibe código de otro A, se requiere la total certeza de que el código resultante de A esté léxica y sintácticamente correcto, además de que se debe contar en B con un mecanismo que aproveche esta característica.

Uno de los casos donde aporta mayor utilidad el renunciar a etapas de compilación, ya sea el análisis léxico o sintáctico, es en los Lenguajes Visuales, dado que para el almacenamiento de un programa se emplea un código intermedio con formato textual para poder realizar los análisis y la traducción a código objeto. Consecuente con lo anterior, es que si se efectua un análisis sintáctico analizando los íconos desde una gramática para lenguajes visuales (tarea de sobra costosa) posteriormente al realizar el proceso de compilación desde el programa almacenado textualmente, se deberá efectuar de nuevo el análisis sintáctico y después el resto de los análisis y procedimientos. Uno de los mecanismos existentes más difundidos para realizar el análisis sintáctico de los Lenguajes Visuales es el presentado en [2], consiste en un mecanismo para verificar la sintaxis de un programa visual mediante una forma especial del conocido análisis LR(0), con los inconvenientes que éste pueda tener, como el requerimiento de la construcción de una tabla de análisis sintáctico (que resulta ser bastante grande como se puede apreciar en el artículo). De otro lado, si no se tome en la cuenta las características operativas del análisis sintáctico de un programa visual, resta la dificultad de extraer cada uno de los sintagmas[3] presentes.

Surgen entonces, dos interrogantes:
¿Cómo efectuar el análisis sintáctico en el código visual sin tanto costo?
¿Cómo implementar procesos de compilación como la traducción desde un programa visual hacia algún tipo de código textual sin repetir el análisis sintáctico?

En las siguientes secciones se propondrán soluciones a estas preguntas. De esta manera, los aportes fundamentales del trabajo presentado son: Primero, una nueva forma para hacer el análisis sintáctico de los Lenguajes Visuales y segundo, la presentación de un caso real de utilización del mecanismo propuesto.

El presente artículo tratará el siguiende orden de ideas: 1. Introducción del artículo, 2. Presentación de la manera de comprobación sintáctica planteada, 3. Exhibición de un ejemplo en el que se usa la nueva forma de análisis sintáctico, 4. Una descripción de la forma como se implementó el editor y 5. Discusión de las conclusiones de la investigación y el desarrollo.

2. Alternativa para comprobación sintáctica de Lenguajes Visuales

Independiente de la especificación gramatical que se utilice la comprobación sintáctica tiene dos niveles: Sintaxis independiente del contexto y Sintaxis dependiente del contexto.

[1] Un compilador se denomina $cruzado$ cuando recibe el lenguaje fuente F, corre sobre una máquina que ejecuta código I y produce un código objeto O para otra máquina diferente a la que se emplea para ejecutar el compilador. Lo anterior se abrevia con la expresión $F_I O$ y se dibuja como: 

.

[2] Se denominan compiladores $escalonados$ aquellos procesadores de lenguajes que toman un par de compiladores $cruzados$ para obtener otro compilador con características diferentes.

[3] Combinación ordenada de significantes que interactúan formando un todo con sentido, dentro de un conjunto de reglas y convenciones sintácticas.

Según la jerarquía de Noam Chomsky, para especificar la sintaxis independiente de contexto de un lenguaje textual, se requiere a lo sumo, una gramática tipo 2, mientras que para especificar la sintaxis dependiente de contexto se necesita mínimo, una gramática tipo 1. De lo anterior, se puede concluir que desde el punto de vista computacional es más costosa la comprobación sintáctica dependiente del contexto que la independiente. Es por este motivo, que en la práctica se prefiere diseñar una gramática tipo 2 ó 3 (para tratar la sintaxis independiente de contexto y dejar el tratamiento de la dependiente en otras etapas posteriores), que construir una gramática tipo 1 (donde se contemplen todas las características posibles desde la sintaxis) y enfrentar un problema mucho más complicado. Esta metodología de dividir y conquistar sería muy útil en el tratamiento de Lenguajes Visuales si se pudieran abstraer y modelar las características independientes de contexto, dentro del análisis en una primera etapa desde el Lenguaje Visual y hacer que se conserven las características de sintáxis dependiente de contexto en el lenguaje textual de almacenamiento (de los programas inicialmente visuales), para su posterior análisis y tratamiento. Lo anterior, requiere que desde la gramática podamos obtener las características independientes de contexto, para esto se emplearán las Gramáticas de Sistemas de Íconos Generalizados, G_sig.

2.1. Introducción a las Gsig (Gramáticas de Sistemas de Íconos Generalizados)

Dentro de los principales problemas que se deben enfrentar al trabajar con lenguajes visuales se encuentra el establecimiento de una especificación gramatical, para así, determinar un mecanismo de traducción a lenguaje objeto y un mecanismo de almacenamiento de código intermedio. En otras palabras, requerimos guardar en algún medio los programas visuales siguiendo las reglas de alguna especificación gramatical, para posteriormente, mediante un mecanismo de traducción, obtener un código objeto.

En [3] se presenta la forma de especificación gramatical denominada Gsig, la cual consiste en la representación textual de la información visual de un determinado constructor o conjunto de constructores. Ampliando la idea, una Gsig almacena en una expresión textual, la información visual de cada constructor independiente X (referida como la parte física)[4], las relaciones con otros constructores del lenguaje (sintagmas llamados la parte lógica) y su respectiva etiqueta, organizadas como se presenta en la ecuación 1.

$$\mathcal{E}[\![X]\!]^{a} = S^{b}\ Sd_1{}^{c}\ \overbrace{(x-y)\ Path(X)}^{Parte\ Física}\ \sim^{d}\ \overbrace{Name(X)}^{Etiqueta}\ :^{e}\ \overbrace{\mathcal{E}[\![Ex(X)]\!]}^{Parte\ Lógica}\ Sd_2{}^{c} \tag{1}$$

[a] Función semántica que nos entrega la especificación del constructor visual X
[b] Símbolo de identificación que permite determinar la especificación de manera única.
[c] Componentes léxicos de delimitación de la especificación
[d] Símbolo de terminación de la Parte Física.
[e] Símbolo de terminación de la Etiqueta.

De forma similar, para el caso de la especificación de un conjunto de constructores visuales, una Gsig almacena en una expresión textual, la especificación de cada uno de los constructores y los correspondientes símbolos de sincronización $\widehat{Sc_i}$, como se muestra en la ecuación 2.

$$\mathcal{E}[\![X_1, \cdots, X_n]\!] = Sd_1\ \mathcal{E}[\![X_1]\!]\ \widehat{Sc_1} \cdots \widehat{Sc_{n-1}}\ \mathcal{E}[\![X_n]\!]\ Sd_2 \tag{2}$$

2.2. Definición formal del nuevo mecanismo de análisis sintáctico para Lenguajes Visuales

Antes de continuar se necesita una definición.

[4] Partes que componen los íconos generalizados presentados por Chang en [1]

Definición 1. *El conjunto* **PRIMEROS_ÍCONOS** *consta de los primeros íconos que se puedan generar de una producción.*

$$\textit{PRIMEROS\_ÍCONOS}(X) = \begin{cases} imagen_X & \textit{Si } X \rightarrow S'\ Sd_1\ (X - Y)\ imagen_X\ \sim\ \ldots, \\ \textit{PRIMEROS\_ÍCONOS}(PL_X{}^a) & \textit{Si } X \rightarrow S'\ Sd_1\ \epsilon\ \sim\ Et_X{}^b\ :\ PL_X\ Sd_2, \\ \bigcup \textit{PRIMEROS\_ÍCONOS}(X_i) & \textit{Si } X \rightarrow S'\ Sd_1\ X_1\ \widehat{Sc_1}\ \cdots\ \widehat{Sc_{n-1}}\ X_n\ Sd_2, \\ \textit{PRIMEROS\_ÍCONOS}(Y \veebar Z)^c & \textit{Si } X \rightarrow Y \mid Z. \end{cases} \tag{3}$$

[a] Parte Lógica del constructor X
[b] Etiqueta dada por el usuario al constructor X
[c] O exclusivo entre las producciones Y y Z

Ya que las Gramáticas de Sistemas de Íconos Generalizados tienen una estructura que las clasifica dentro de la jerarquía de Noam Chomsky como de tipo 2, podemos afirmar que cada producción nos determina unas características independientes del contexto así:

- Características independientes de contexto en las producciones de los operadores independientes:

$$X \rightarrow S'\ Sd_1\ ParteFísica_X\ \sim\ Etiqueta_X\ :\ \underbrace{ParteLógica_X}_{LX}\ Sd_2 \tag{4}$$

  1. Los símbolos de sincronización: $S',\ Sd_1,\ \sim$ y **:** . Están determinados para cada constructor independiente del lenguaje.
  2. La $ParteFísica$ de cada constructor está determinada por la aplicación de edición de programas visuales.
  3. La ventana resultado de la $expansión$ del ícono etiquetado con $Etiqueta_X$ sólo podrá contener íconos que pertenezcan a PRIMEROS_ÍCONOS(LX).

- Características independientes de contexto en las producciones de los conjuntos de constructores dentro de una $expansión$:

$$LX \rightarrow S'\ Sd_1\ X_1\ \widehat{Sc_1}\ \cdots\ \widehat{Sc_{n-1}}\ X_n\ Sd_2 \tag{5}$$

  1. Los símbolos de sincronización: $S'\ Sd_1,\ Sd_2\ \widehat{Sc_1}$ ... $\widehat{Sc_{n-1}}$. Están determinados para cada conjunto de constructores dentro de una $expansión$.
  2. La ventana LX únicamente puede contener íconos que pertenezcan a PRIMEROS_ÍCONOS(LX).

Se necesita un par de definiciones para proseguir.

Definición 2. *En adelante, llamaremos* **Ventana de Expansión** *al área de trabajo resultande de* $expandir$ *un ícono en alguna aplicación para edición de programas visuales y donde se pueden desplazar íconos de contructores del lenguaje. De igual forma, una Ventana de Expansión* **Activa** *es una Ventana de Expansión que tiene el enfoque en ese instante, se representa por una ventana con los bordes más gruesos (ver figura 2).*

Definición 3. *En lo subsiguiente se hará referencia como* **Ventana de Constructores Activos** *al repositorio de íconos de constructores del lenguaje (presente en una aplicación para la edición de programas visuales) que pueden ser desplazados hacia su respectiva Ventana de expansión activa para su utilización. Se muestra como una ventana con los íconos organizados en vertical (ver figura 2).*

Consecuente con las definiciones 2 y 3 se puede definir qué constructores pueden intervenir en la parte lógica de cada constructor visual expresado con una Gsig.

Definición 4. *Los* **Íconos Activos** *son las gráficas que aparecerán en una ventana de constructores activos y que podrán ser desplazados hacia una ventana de expansión determinada.*

Si X es un constructor visual, entonces en su Ventana de Expansión (cuando esté Activa) sólo se podrán desplazar los respectivos íconos activos, los cuales corresponden a PRIMEROS_ÍCONOS(LX).

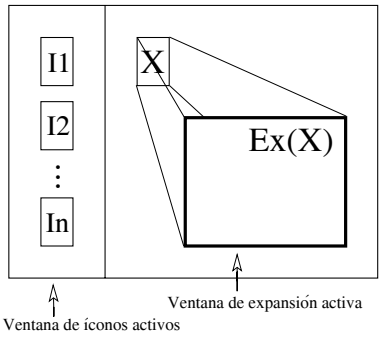

Figura 2: Ventana de constructores activos y Ventana de expansión activa.

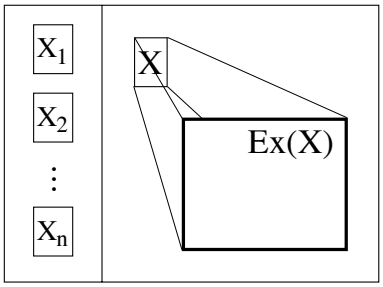

Figura 3: Íconos permitidos en la ventana de expansión activa del ícono del constructor X.

$$X \rightarrow S'\ Sd_1\ ParteFísica \sim Etiqueta_X : \underbrace{ParteLógica}_{LX}\ Sd_2$$

$$LX \rightarrow S'\ Sd_1\ X_1\ \widehat{Sc_1}\ \cdots\ \widehat{Sc_{n-1}}\ X_n\ Sd_2$$

$$X_1 \rightarrow S'\ Sd_1\ (x-y)\ imagen_{X_1} \sim \cdots$$
$$\vdots$$
$$X_n \rightarrow S'\ Sd_1\ (x-y)\ imagen_{X_n} \sim \cdots$$

$$\textit{PRIMEROS\_ÍCONOS}(LX) = \{imagen_1,\ \ldots,\ imagen_n\}$$

$$imagen_{X_i} \stackrel{\text{def}}{=} \boxed{\text{Xi}}$$

De esta manera, se puede restringir la utilización del conjunto de íconos en cada ventana del editor de programas visuales asegurando que en cada ventana de expansión (parte lógica del constructor respectivo) solamente se permitan desplazar los íconos de constructores adecuados, logrando de esta forma, que todo programa editado en un entorno que contenga y emplee la definición de las ventanas de constructores activos será gramaticálmente correcto con respecto a su sintaxis independiente del contexto. Pudiendo así, generar el programa en código textual de almacenamiento sin errores sintácticos independientes del contexto, para analizarlo en función de la sintaxis dependiente del contexto.

De esta forma, se ha dividido el análisis sintáctico de un lenguaje visual en dos etapas diferentes:

1. Análisis sintáctico independiente de contexto: que se efectúa al momento de la edición del programa visual no permitiendo que el usuario pueda desplazar un ícono a una ventana que no le corresponde o, en otras pala-

bras, se restringen los íconos activos dependiendo de la ventana de expansión activa, en términos de lingüística computacional, se acota el conjunto de lexemas que pueden aparecer en un contexto (se define un paradigma[5]).

2. Análisis sintáctico sensible al contexto: Se efectúa (en las etapas de análisis semántico y traducción) sobre el modelo textual de almacenamiento con la precondición de que está correcto con respecto a la sintáxis independiente del contexto. Lo anterior, se consigue empleando un mecanismo de traducción como el T_Gsig [4], además de la inclusión de tablas de símbolos, (para un recorrido eficiente se emplean Tablas Hash) en donde se almacena información de: identificadores (ámbito, clase de identificador), declaraciones de ámbito (lista de identificadores de variables y métodos), envío de mensajes (lista de identificadores), métodos (lista de parámetros), para recorrer cada una de las tablas y verificar las condiciones dependientes del contexto como: todas las variables debe tener un valor asociado, la cantidad de argumentos en un método deben ser igual a la cantidad de parámetros declarados, se deben emplear métodos existentes. Lo anterior, a sabiendas de que el código recibido cumple con las características sintácticas independientes del contexto.

3. Ejemplo: El editor del Cálculo Visual GraPiCO, E_GraPiCO

Ante el requerimiento de una aplicación que permita la programación en Calculo GraPiCO [5] del Grupo de Invetigación AVISPA[6] se diseñó e implementó un editor visual denomidado E_GraPiCO; para efectuar su parser se empleó el mecanismo de comprobación sintáctica propuesto en este artículo. En las secciones posteriores se hace su introducción.

3.1. Alfabeto de GraPiCO

El la figura 4 se ilustran los íconos que pueden a parecer en un momento dado dentro de un programa GraPiCO (Ventana de Íconos Activos más general) y por medio de glosas se explican cada uno de los elementos gramaticales utilizados en E_GraPiCO, esta también es una forma de presentar el grupo de Íconos Activos de cada constructor que genere una Ventana de Expansión

3.2. Descripción del área de trabajo (Características del editor)

Los lenguajes visuales están basados en un lenguaje icónico que representa los elementos del problema en discusión. Por tanto se propuso crear una interfaz que permita construir programas gráficamente mediante un alfabeto gráfico definido, facilitándole al programador representar en un área de diseño las diferentes instrucciones GraPiCO.

Para ello, la interfaz de desarrollo ofrecerá un entorno amigable para el programador, con las respectivas herramientas gráficas necesarias para la total representación de las sentencias del cálculo GraPiCO. Las herramientas deben contener las características indispensables para la comodidad del usuario durante el desarrollo de sus programas; entre estas características encontramos:

- La validación sintáctica definiendo la disponibilidad de las herramientas según el tipo de sentencia que se esté programando.
- El control de las ayudas contextuales y elementos gráficos de soporte al usuario.
- La navegabilidad entre ventanas para describir la estructura del programa GraPiCO.
- Barras de herramientas, que permiten ejecutar funcionalidades muy repetidamente.

3.3. Representación de un Programa GraPiCO

Se desarrolló un entorno visual que le brinda al programador las herramientas gráficas necesarias para diagramar sobre una Lista de Ambientes (área de trabajo), obteniendo como resultado un Programa GraPiCO. Un Programa GraPiCO se compone de una Lista de Ambientes y a su vez cada Lista de Ambientes puede tener una Lista de Constructores; un Ambiente es el espacio donde el programador dibujará las instrucciones que constituyen su Programa (formalmente una Ventana de Expansión). Donde cada Ambiente utilizado tiene definido un Conjunto de Gráficos que se relacionan entre sí para expresar una instrucción, y a su vez son el punto de partida para generar nuevos Ambientes.

[5] Conjunto virtual de elementos de una misma clase gramatical, que pueden aparecer en un mismo contexto.

[6] **A**mbientes **Vis**uales de **P**rogramación **A**plicativa, Universidad Javeriana, Universidad Del Valle, Universidad De Los Andres, Instituto IRCAM

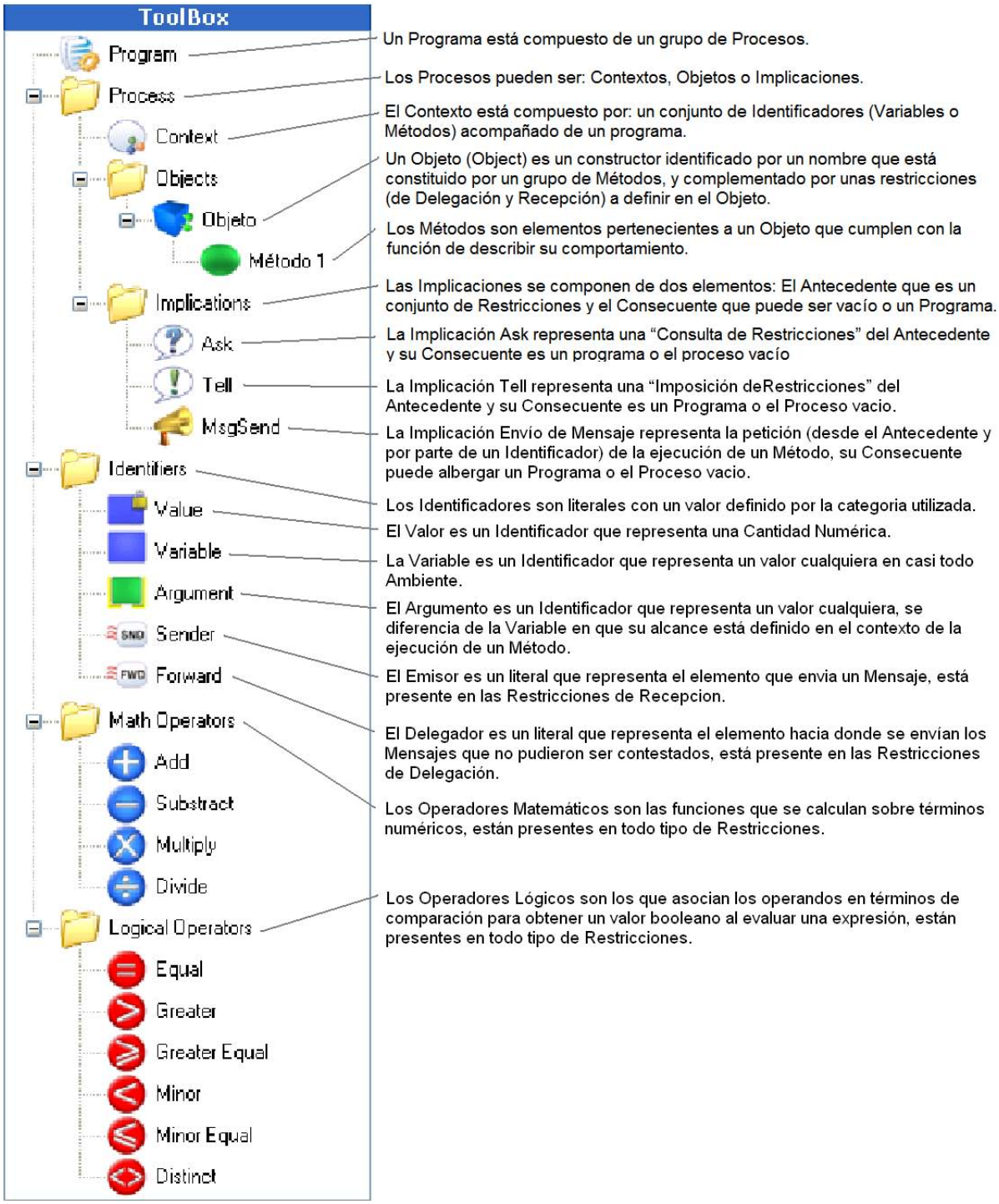

Figura 4: Representación del lenguaje icónico manejado a partir de la gramática del Cálculo Visual GraPiCO.

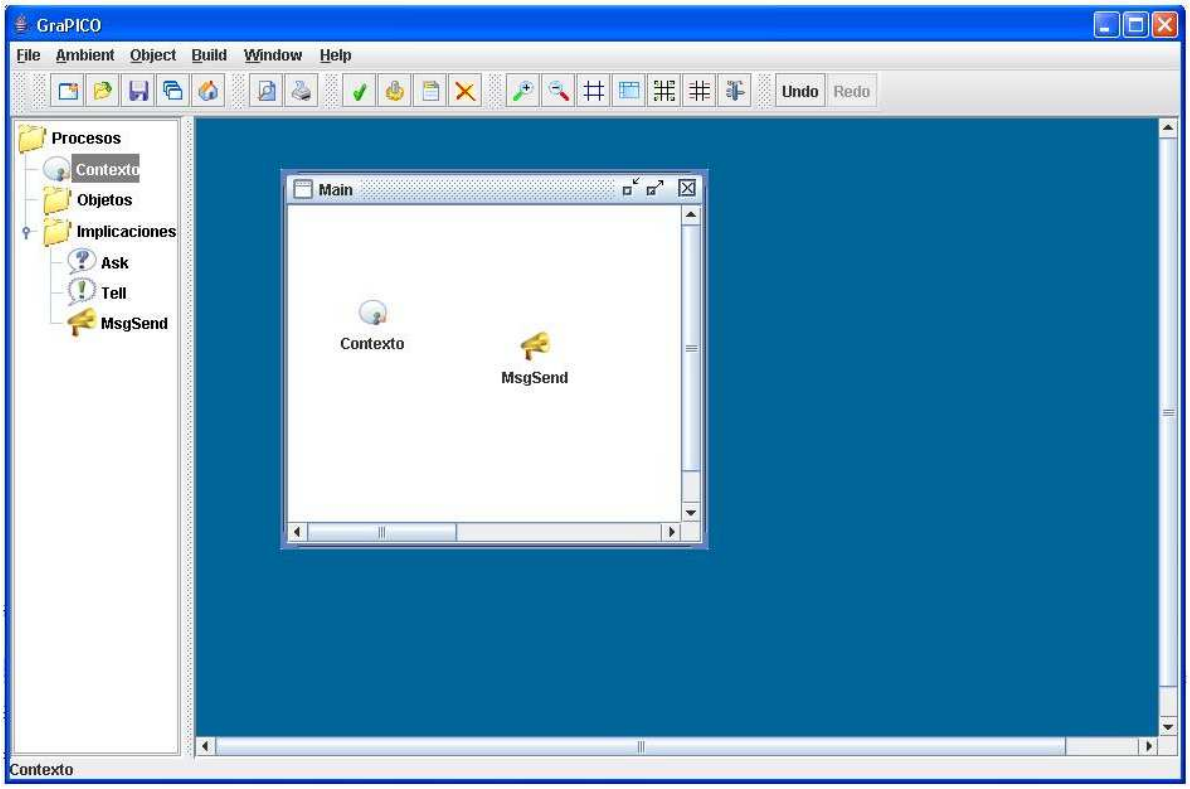

Figura 5: Diseño del formulario Principal

Los objetos definidos pueden ser usados como procesos en distintos Ambientes dependiendo de las Restricciones Sintácticas visuales impuestas[7].

3.4. Componentes del entorno gráfico

El entorno debe proporcionar los mecanismos necesarios para que el programador dibuje de manera cómoda las instrucciones de su Programa, utilizando las herramientas gráficas, sobre los Ambientes o Paneles. Los mecanismos ofrecidos por E_GraPiCO son:

- Menú: Despliega un listado de las opciones disponibles del entorno gráfico de acuerdo a las categorías en que las opciones se encuentren agrupadas.
- Barra de Herramientas: Despliega una serie de íconos que corresponden a algunas herramientas utilizadas para tareas que proporciona el entorno, las cuales son de uso frecuente por el programador de Cálculo Visual GraPiCO.
- Caja de herramientas (Ventana de Íconos Activos): Despliega organizadas en un árbol de jerarquías, las herramientas disponibles para la construcción de las sentencias del Cálculo Visual GraPiCO, dependiendo de las Restricciones de cada Ambiente.
- Área de construcción del programa: En esta área serán organizados los paneles de diseño para la construcción del programa. Esta área es la que contendrá todas las ventanas, donde cada ventana representa un Ambiente y éste a su vez una instrucción GraPiCO.
- Formulario principal: Es la ventana que integra todos los elementos mencionados anteriormente.

[7] Definición de los conjuntos **Íconos Activos** para cada Ambiente

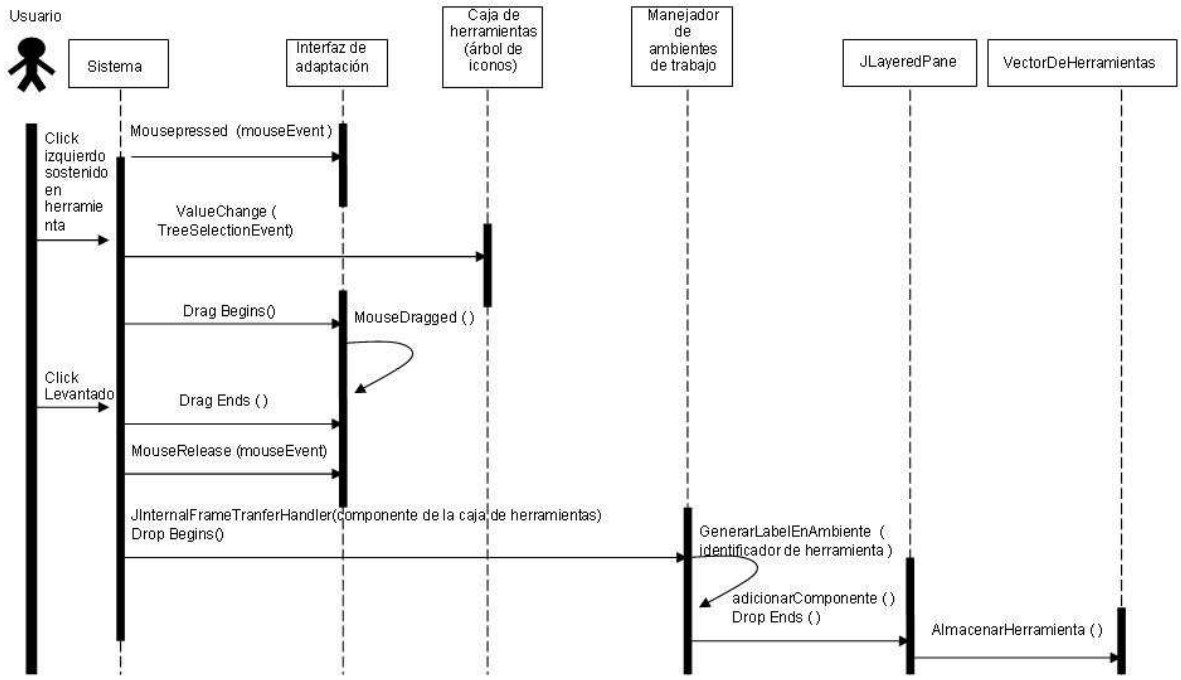

Figura 6: Diagrama de secuencia DnD.

Para dibujar una sentencia en un programa GraPiCO, el programador debe seleccionar de la caja de herramientas un ícono que representa un elemento de la gramática utilizada, arrastrarlo y soltarlo sobre el Ambiente o ventana actual del área de construcción. Cuando un ícono es soltado sobre un Ambiente se debe interpretar sus características y dibujarlo sobre el Ambiente en donde indique la posición del puntero del Mouse.

Para mostrar la manera como se comporta la aplicación se presentarán los diagramas de secuencias más relevantes en los apartados siguientes.

3.5. Diagrama de secuencia Drag And Drop (DnD) de la Caja de Herramientas hacia los ambientes de trabajo

El diagrama de la figura 6 presenta el manejo del desplazamiento de los constructores de la caja de herramientas (Ventana de Íconos Activos) hacia el ambiente (Ventana de Expansión Activa).

3.6. Diagrama de secuencia de activación de un Ambiente y actualización de la Caja de Herramientas

La figura 7 contiene el diagrama que muestra como se efectúan los cambios en la caja de herramientas (conjunto de Constructores que aparecen en la Ventana de Iconos Activos) dependiendo del ambiente (Ventana de Expansión) activo.

4. Descripción del Desarrollo del editor

4.1. Herramientas para el desarrollo del proyecto

Como herramienta, en el lenguaje de programación JAVA, existe el entorno de desarrollo NETBEANS IDE basado en el software de NETBEANS.org que proporciona mecanismos para escribir, diseñar, compilar, depurar y ejecutar programas JAVA ayudando en la generación de código, documentación (Developer collaboration) y de clases que hacen énfasis en el manejo de API, como por ejemplo, Swing (Grupo de componentes escritos en java para diseñar interfases graficas de usuario multiplataforma), JAVA 2D (incorporación de gráficos 2D de alta calidad, texto e imágenes) y, soporte Drag and Drop (arrastrar y soltar: trasferencia de objetos gráficos entre paneles y aplicaciones).

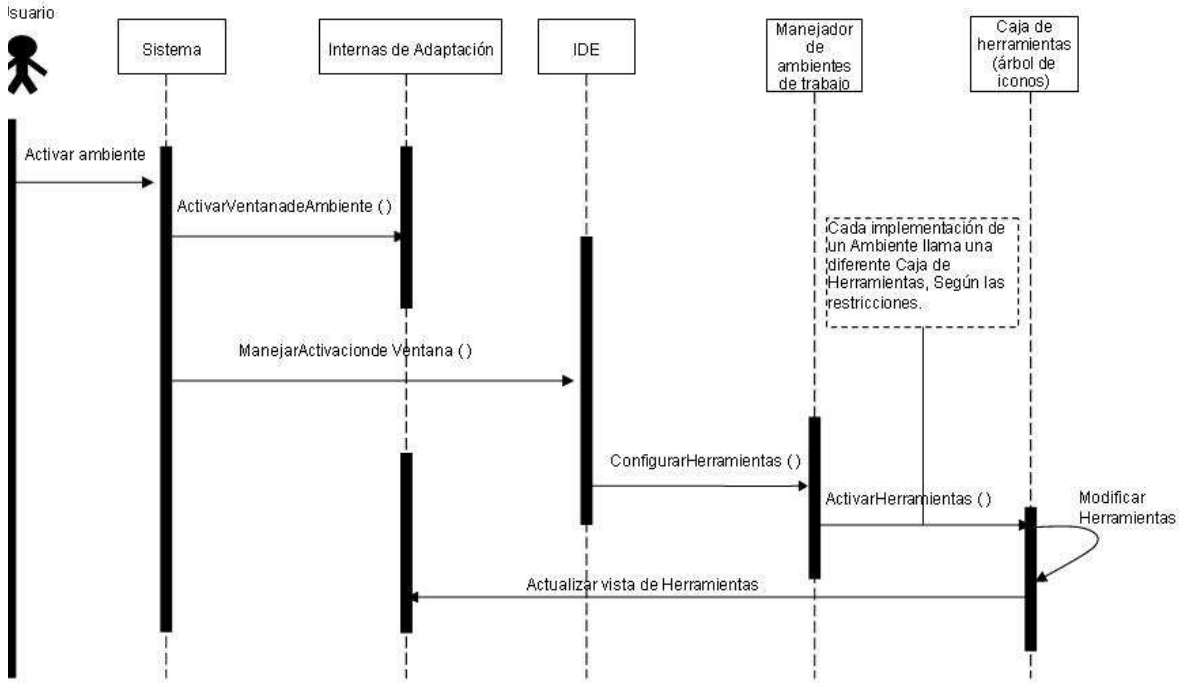

Figura 7: Diagrama de secuencia de activación y actualización.

4.2. Funcionalidades del Editor Gráfico para El Cálculo Visual GraPiCO.

4.2.1. Análisis Sintáctico Dinámico

Uno de los avances del editor, es la propiedad de realizar un análisis sintáctico dinámico, cuya función es analizar la manera en que el usuario está utilizando el lenguaje icónico del Cálculo Visual GraPiCO y que concuerde con los términos de la gramática expuestos en [5], mientras se lleva a cabo conjuntamente la creación de él, concluyendo en que todas las listas de Ambientes generadas en el Programa hecho por el usuario, se ejecute sin errores sintácticos (de orden) a partir de reglas sintácticas creadas que convierten el programa visual con cada unos de sus elementos gráficos a una forma textual, con el objetivo de entregarle a las siguientes etapas de compilación un programa almacenado de forma textual sintácticamente correcto y que el compilador no tenga que realizar un análisis sintáctico. El Análisis Sintáctico Dinámico está implícito en el editor, haciéndolo dinámico debido a que el usuario no puede cometer errores de programación.

Para utilizar la teoría expuesta en la sección 2.2 la implementación del entorno gráfico de E_GraPiCO se diseñó de manera que la Ventana de Constructores Activos (planteada en la Definición 3) aparece representada por la caja de herramientas (ToolBox en el desarrollo), donde se situan los dibujos con los que se puede construir los programas GraPiCO (el conjunto *PRIMEROS_ÍCONOS(X)* introducido en la Definición 1) mediante su desplazamiento hacia las ventanas en el Área de Construcción de Programa o Desktop (estas ventanas son denominadas Ambients, tipo específico de Frame en la implementación para la construcción de sentencias de lenguaje); la conformación del grupo de íconos está condicionada por la ventana que en ese instante tenga el enfoque (mostrada como Ventana de Expansión Activa en la Definición 2), así, cuando un determinado ícono es expandido, su correspondiente conjunto de Íconos Activos (según la Definición 4) es dispuesto en la caja de herramientas para hacer posible su empleo en la edición de programas.

Para desarrollar este análisis es necesario lograr que cada ícono de constructor del lenguaje active su respectiva caja de herramientas. Esta relación entre constructores y conjunto de Íconos Activos en la caja de herramientas se puede modelar mediante una tabla de reglas de disponibilidad de herramientas para los Ambients como la que se utilizó para E_GraPiCO y que se presenta en la figura 9. Un ejemplo para demostrar esta funcionalidad, es cuando el usuario se encuentra en un Ambient Método, en donde sólo podrá desplazar de la caja de herramientas los constructores de Programa, Proceso e Identificadores. La operabilidad de estas reglas en el editor es exhibida en el diagrama de secuencia en la figura 8.

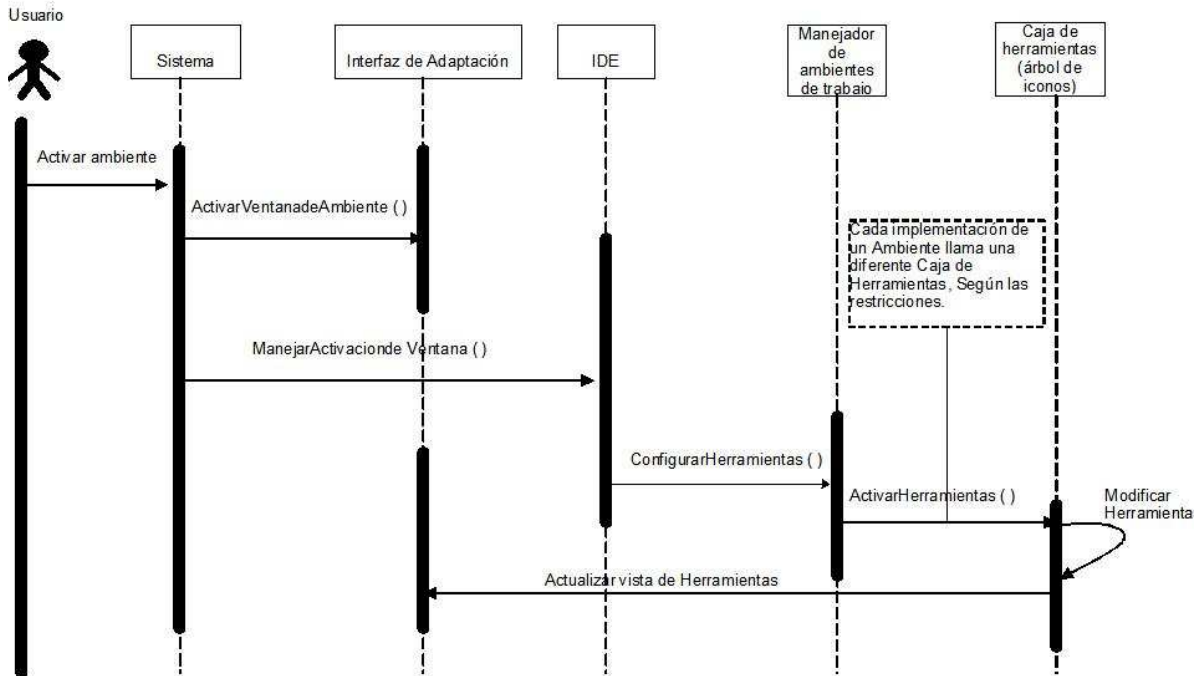

Figura 8: Diagrama de secuencia de actualización de Ambients y actualización de la caja de herramientas.

Constructor \ Ambients	Prog.	Context	Method	Constraints	Constraints Group	Msg Send
Program		X	X			
Process	X	X	X			
Method		X				X
Math Operators				X		
Logical Operators				X		
Identifiers		X	X	X		X
Constraints					X	

Figura 9: Tabla de la relación entre constructores y ventanas (Ambients)

4.2.2. Almacenamiento Textual de los Programas GraPiCO

Frente a la necesidad de brindar la posibilidad de guardar un programa para su posterior edición o ejecución se empleó una representación textual de cada constructor utilizado en el Cálculo Visual GraPiCO por medio de las G_sig presentadas en [3], de manera que el archivo almacenado conserve la jerarquía generada en los programas. Es decir, a partir del Ambient principal (conocido en el editor como Main), se crea un orden para el conjunto de contructores utilizados siguiendo el árbol sintáctico del programa.

5. Conclusion

De esta manera, mediante el mecanismo de $dividir\ y\ conquistar$ se ha reducido la dificultad del análisis sintáctico de un programa visual, pues en lugar de hacer el análisis sintático en su totalidad desde el programa visual, se plantea una primera etapa de análisis sintáctico independiente de contexto de manera dinámica, y se efectúa el análisis dependiente de contexto de manera estática, es decir, se abstraen las comprobaciones sintáticas que se pueden hacer al momento de la edición (sin mucho costo computacional) de las que requieren mayor tratamiento y almacenamiento de información, de otra forma, el análisis dependiente de contexto.

Gracias a los métodos formales que se emplearon para presentar el nuevo mecanismo es posible la demostración de correción de la etapa de análisis sintáctico a un lenguaje de programación.

por medio de la implementación de E_GraPiCO se pudo constatar que, en efecto, la nueva forma de análisis sintactico introducida es útil y se puede ayudar en la elaboración de nuevos lenguajes de programación visual donde se requiera un parsing agil y correcto.

Referencias

[1] CHANG, S. In *Principles of Visual Programming Systems.* (1990).

[2] COSTAGLIOLA, G., AND POLESE, G. Extended positional grammars. In *Tecnical report, Dipatimento di Matematica ed Informatica, University of Salerno, Salerno, Italy* (2001).

[3] TAVERA, C., AND DÍAZ, J. Alternativa para especificación sintáctica: Gramática de sistemas de íconos generalizados: Gsig. In *Congreso Argentino de Ciencias de la Computación, San Luis, Argentina.* (2006).

[4] TAVERA, C., AND DÍAZ, J. Alternativa de mecanismo de traducción para lenguajes visuales: T_gsig. In *Reporte técnico; http://eisc.univalle.edu.co/∼catavera* (2007).

[5] TAVERA, C., AND DÍAZ, J. Nuevo cálculo visual: Grapico. In *Congreso Colombiano de Computación, Universidad Javeriana, Bogotá, Colombia* (2007).