INGENIERÍA Y DESARROLLO

ARTÍCULO CIENTÍFICO / RESEARCH ARTICLE

Alternativa de mecanismo de traducción de lenguajes mediante análisis de símbolos de sincronización: T_Gsig

Alternative language translation mechanism by means of synchronization-symbol analysis: T_Gsig

Carlos A. Tavera*
Juan F. Díaz**

* PhD. Ciencias de la Computación. Profesor Titular de la Universidad de San Buenaventura, Cali (Colombia). catavera@usbcali.edu.co

Correspondencia: Av. 10 de mayo, La Umbría, vía a Pance. A.A. 7154 y 25162. Cali (Colombia).

** PhD. Informática. Profesor Titular de la Universidad del Valle, Cali (Colombia). jdiaz@univalle.edu.co

Número 26
Julio-diciembre, 2009
ISSN: 0122-3461

Resumen

En este artículo se propone un mecanismo de traducción que transforma un programa visual especificado sintácticamente en forma textual, hacia un código de un lenguaje textual. Fundamentalmente, el mecanismo consiste en tomar la especificación de un constructor simple o compuesto, eliminar la información visual mediante análisis de símbolos de sincronización, y transformarlo en un programa equivalente en lenguaje textual. Inicialmente, en el artículo se presenta de manera formal el mecanismo de traducción, luego, se muestran las plantillas para su implementación, y al final, se presentan los resultados de la traducción a través de reglas.
Palabras clave: Lenguajes visuales, Traducción de lenguajes.

Fecha de recepción: 2 de septiembre de 2008
Fecha de aceptación: 24 de marzo de 2009

Abstract

In this paper is proposed a mechanism of translation that transforms a visual language syntactically specified by textual form, into code of a textual language. Fundamentally, the mechanism consist in taking the simple or composite constructor specification, eliminate the visual information through analysis of synchronization symbols, and transform it into a equivalent program in textual language. Initially, the translation mechanism is present formally, later, the templates for its implementation are shown, and at the end, translation results are presented by rules.
Keywords: Visual languages, language translation.

1. INTRODUCCIÓN

Hacia el año de 1998 el grupo de investigación AVISPA (**A**mbientes **VIS**uales de **P**rogramación **A**plicativa) crea el lenguaje CORDIAL, un lenguaje visual para un subconjunto del cálculo PiCO [1]. Luego de las pruebas de eficiencia que el grupo AVISPA le practicó a CORDIAL, se pudo comprobar que aunque el producto de la compilación calculaba los resultados con éxito, la velocidad de ejecución en la mayoría de los casos no fue aceptable.

Más tarde, se logró establecer que el problema estaba, particularmente, en el código generado que, en ciertos casos, contenía código duplicado, inactivo o innecesario. Consecuente con lo anterior, el grupo AVISPA inicia el proyecto del nuevo Cálculo GraPiCO [2], un cálculo visual para la totalidad del cálculo PiCO, con la utilización de un mecanismo de especificación textual, llamado Gsig (introducido más adelante), y un nuevo mecanismo de traducción denominado T_Gsig, el tema central de éste artículo.

1.1 Estado del arte y al aporte de la investigación

El tratamiento formal de código textual es un tema que ha persistido desde los inicios de la Ciencia de la Computación. Junto con internet y la necesidad de almacenar y manipular la información contenida en ella, se origina lo que ahora llamamos los lenguajes de marcación[1][3]; dentro de los más famosos se encuentran el HTML (Hyper Text Markup Language)[4] para la descripción de páginas web, el OWL (Web Ontology Language)[5] para presentación de ontologías y los esquemas XML para marcado descriptivo (Extensible Markup laguage)[6]. Hasta el momento se han empleado los lenguajes de marcación para tratamiento (ver [7]) y transformación de código textual, como el trabajo en [8]; desde el punto de vista gráfico, también se han usado para representar y traducir hojas de cálculo, como el que se presentó en [9], pero los programas visuales requieren un poder expresivo más grande y una forma eficiente de traducción, pues los costos computacionales de los lenguajes icónicos pueden ser grandes (un ejemplo se muestra en [10]) debido a la complejidad y la extensión del código textual requerido para modelar un programa visual.

El aporte de este documento es proponer una manera de efectuar la traducción de representaciones textuales de código visual de forma general, simple, formal e implementable.

1.2 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 lugar los programas visuales siguiendo unas reglas sintácticas, para posteriormente, mediante un mecanismo de traducción, obtener un código objeto. En [11] se presenta la forma de especificación gramatical denominada Gsig, la cual consiste en la representación

[1] Lenguajes de marcación: es una forma de codificar un documento que, junto con el texto, incorpora etiquetas o marcas que contienen información adicional de la estructura del texto o su presentación. World Wide Web Consortion.

textual de la información visual de un determinado Constructor Simple[2] o Compuesto[3]. Ampliando la idea, una Gsig almacena en una expresión textual, la información visual de cada constructor simple *X* empleando los conceptos de íconos generalizados[4] presentados en [12] y su respectiva etiqueta, organizados como se presenta en la ecuación 1. Donde *T* representa la función semántica que nos entrega la especificación del constructor visual *X*; internamente, esta especificación está delimitada por los símbolos Sd_1 y Sd_2 y está identificada de manera única por el símbolo de encabezamiento S'.

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

De forma similar, para el caso de la especificación de un constructor compuesto, una Gsig almacena en una expresión textual, la especificación de cada uno de los constructores simples que lo componen y los correspondientes símbolos de sincronización Sc_i, como se muestra en la ecuación 2.

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

1.3 **Introducción al T_Gsig**

Con lo anterior, se puede observar que, dado que para la fase de traducción se requiere fundamentalmente la información contenida en la parte lógica –como está construida la gráfica al interior–, la traducción de un programa total o un constructor simple en particular dentro de un programa, especificados a través de Gsig, consiste de la separación de la parte lógica y su posterior traducción interna; de otro lado, la traducción de un constructor compuesto consiste de la traducción de cada uno de los constructores que hacen parte del conjunto. En éste artículo se presenta T_Gsig, un mecanismo de traducción de la especificación de un lenguaje visual utilizando Gsig en un lenguaje textual mediante la técnica de "scanning" (análisis léxico) y no a través de "parsing" (análisis sintáctico) como es presentado en [13], lo cual

---

[2] Conjunto de Constructores Simples.

[3] Ícono en particular sin tomar en la cuenta su entorno.

[4] "representaciones duales de objetos consistentes de una parte lógica (el significado, o lo que es lo mismo, la definición de su estructura interna) y una parte física (información que relaciona los constructores desde el punto de vista netamente visual: localización y los dibujos)". Convergencia de definiciones en [12] y [10].

permite una implementación generalizable mediante el diseño orientado al objeto y más eficiente.

2. DEFINICIÓN DEL MECANISMO DE TRADUCCIÓN

Acorde con la definición de las Gsig, la traducción (expresada mediante T[5]) de un programa generado con una Gsig, hacia uno de constructores netamente textuales, se contempla en los casos siguientes:

Caso 1: Traducción de un constructor simple que se ha expresado mediante una Gsig:

$$T[\![S'S_{d_1}(x-y)\ Path(X) \sim Name(X) : ParteLógica\ S_{d_2}]\!]$$

$$= T[\![S']\!]\ T[\![S_{d_1}]\!]\ T[\![ParteLógica]\!]\ T[\![S_{d_2}]\!]$$

Caso 2: Traducción de un constructor compuesto especificado a través de una Gsig:

$$T[\![S_{d_1}\ X_1\ S_{c_1}\ \cdots\ S_{c_{n-1}}\ X_n\ S_{d_2}]\!] = T[\![S_{d_1}]\!]\ T[\![X_1]\!]\ T[\![S_{c_1}]\!]\ \cdots T[\![S_{c_{n-1}}]\!]\ T[\![X_n]\!]\ T[\![S_{d_2}]\!]$$

Del Caso 1 se obtiene, que la traducción de un programa generado por una Gsig (código fuente), hacia su respectivo código de lenguaje textual (código objeto), consiste de: a) La supresión de la parte física del código fuente, b) La traducción de la parte lógica del código fuente, y c) La transformación de los símbolos de sincronización del código fuente, en los correspondientes símbolos de sincronización del código objeto.

Lo anterior, es consecuente con la estructura de las gramáticas de sistemas de íconos generalizados - Gsig, dado que en la parte física, es donde quedan almacenados los datos visuales sobre la interconexión del respectivo constructor con el resto, y la parte lógica, contiene la definición de la configuración al interior del constructor.

Gracias a las anteriores observaciones sobre el mecanismo de traducción, se define prácticamente la traducción de cada constructor simple generado

[5] En adelante, emplearemos $T[\![Y]\!]$, como la función semántica que recibe una expresión Y (generada por una G_sig) y entrega su traducción en código de un lenguaje textual

por una Gsig con la estructura presentada en la ecuación 3 y mediante un objeto con el diseño presentado en la figura 1 a) (Donde **Type** se refiere al tipo de datos elegido al momento de desarrollar, para almacenar la especificación mediante una Gsig).

$$X \rightarrow S' S_{d_1} Y \sim Etiqueta : Z S_{d_2} \quad (3)$$

Donde Z es la Parte lógica de X ó z_Gsig, y la representación Gsig de X es x_Gsig.

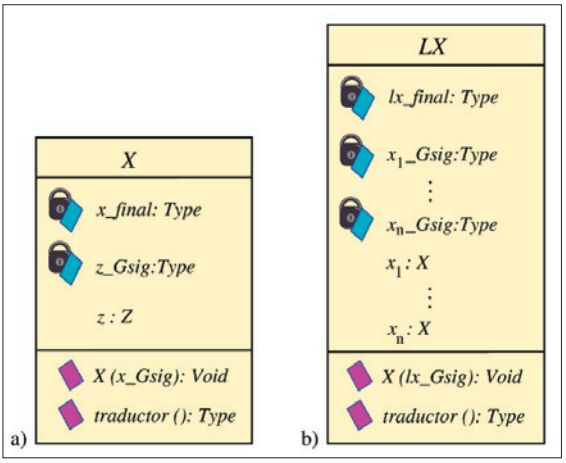

Figura 1. a) Constructor simple X, b) Constructor compuesto LX.

Posterior al diseño del objeto del constructor simple X, se presenta en las figuras 2 y 3 una plantilla para la implementación de la traducción de los constructores simples especificados por una Gsig.

```
//Clase que efectúa la traducción del constructor visual X.
public class X
{
private Type x_final; //Código traducido del constructor X.
private Type z_Gsig; //Parte lógica de X, Z en términos de Gsig.
Z z; //Objeto z como instancia de Z (la parte lógica de X).
```

Figura 2. Implementación de Objeto del constructor X, sección: Atributos.

Ya que el mecanismo de traducción presentado supone un código fuente sin errores sintácticos, esto permite una traducción mucho más eficiente, puesto que la búsqueda de los sintagmas[6] se efectúa mediante la extracción de segmentos de código delimitado por símbolos de sincronización.

De la plantilla presentada en las figuras 2 y 3, se puede observar que la traducción de un constructor simple *X*, inicia con la creación del objeto *x* (instancia de la clase *X*), la cual activa el método constructor, y éste a su vez, emplea la función *separador* (particular al componente *Z*) para la extracción de la parte lógica de la especificación de *X* (guardada en *x_Gsig*) la cual está delimitada por los símbolos de sincronización *:* y Sd_2, y su posterior almacenamiento en *z_Gsig*.

```
/*Constructor de las instancias de X, que recibe al constructor X en
Gsig, posteriormente, se encarga de extraer la parte lógica de X por
medio de separador, y la almacena en z_Gsig.*/

public X (Type x_Gsig){ z_Gsig = separador(x_Gsig);}

//Método que se encarga de traducir el constructor X.
public Type traductor() {
//Creación del objeto z (parte lógica de X), a partir de
//z_Gsig.
z = new Z(z_Gsig);

/*Obtención del código final del constructor X, mediante el ordenamiento
de las traducciones de los símbolos de sincronización S', Sd1, Sd2 y la
traducción de la parte lógica de X, configurada en el objeto z (obtenida,
al igual que con x, mediante su correspondiente método traductor)*/.
x_final=ordenación(T⟦S'⟧, T⟦Sd1⟧, z.traductor(), T⟦Sd2⟧);

return x_final;
                        }
}
```

Figura 3. Implementación de Objeto del constructor *X*, sección: Métodos.

Luego, con la activación del método traductor del objeto *x*, con la información contenida en *z_Gsig* se crea otro objeto *z* (instancia de la clase correspondiente a la parte lógica de *X*), y por ende, se activa el constructor de *z* y su respectivo método *traductor*. De esta manera, sucesivamente se navega el

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

árbol sintáctico de forma virtual, por medio de la jerarquía de clases. Al final, cuando las diferentes partes lógicas ya están traducidas, se procede a su disposición con la función *ordenación*, con el fin de obtener un código que se ajuste a las reglas sintácticas del lenguaje textual objeto. La implementación de las funciones *separador* y *ordenación*, y el método *traductor* depende del lenguaje que se está utilizando para realizar la implementación.

De otro lado, por el Caso 2, se tiene que la traducción de un constructor compuesto especificado textualmente utilizando una Gsig, consiste de: a) Traducción de cada constructor X_i que compone el conjunto, b) Transformación de los símbolos de delimitación Sd_1 y Sd_2, c) Transformación de cada símbolo de sincronización Sc_i, y d) Organización de todas las anteriores traducciones.

Dadas las anteriores observaciones sobre el mecanismo de traducción, se define -para su implementación- la traducción de cada constructor compuesto generado por una Gsig con la estructura presentada en la ecuación 4 y mediante un objeto con el diseño presentado en la figura 1 b).

Después del diseño del objeto del constructor compuesto *LX*, se muestra en las figuras 4 y 5 una plantilla para la implementación de la traducción de los constructores compuestos especificados por una Gsig.

$$LX \rightarrow \overbrace{S_{d_1} \underbrace{X_1}_{\substack{Representación\ Gsig\ de\ X_1 \\ \stackrel{def}{=} x1\_Gsig}} S_{c_1} \cdots S_{c_{n-1}} X_n S_{d_2}}^{Representación\ Gsig\ de\ LX \stackrel{def}{=} lx\_Gsig} \quad (4)$$

```
//Clase que efectúa la traducción de un constructor LX.
public class X
{
private Type lx_final; //Código traducido del constructor LX.

//Constructores de la lista en términos de Gsig.
private Type x1_Gsig, … ,xn_Gsig;

/*Definición de cada objeto xi como instancia de la clase del
constructor Xi respectivo*/
X1 x1; … ; Xn xn;
```

Figura 4. Implementación de Objeto para traducción de *LX*, sección: Atributos.

De igual forma que con la plantilla de las figuras 2 y 3, con la plantilla presentada en las figuras 4 y 5 podemos notar que la traducción de un constructor compuesto *LX* inicia con la creación del objeto *lx* (instancia de la clase *LX*), la cual activa el método constructor, y éste a su vez, emplea la función $separador_i$ para la extracción del segmento de código de cada componente X_i, los cuales están delimitados por sendos símbolos de sincronización Sc_{i-1} y Sc_i (cuando $i \neq 1, n$) , Sd_1 y Sc_1 (cuando $i = 1$) , o Sc_{n-1} y Sd_2 (cuando $i = n$), y su posterior almacenamiento en x_i_*Gsig* respectivamente. Luego, con la activación de cada método $traductor_i$, con la información contenida en cada x_i_*Gsig* se crea el objeto xi (instancia de la clase correspondiente al componente X_i), y por ende, se activa el constructor de cada x_i. Finalmente, cuando los diferentes componentes ya están traducidos, se procede a su arreglo con la función ordenación, con el fin de obtener un código que se ajuste a las normas sintácticas del lenguaje objeto.

En la próxima sección, se demostrará la *validez* de la función de traducción *T* la definición de traducción mediante reglas. Esta lista de reglas, es empleada para transformar un programa GraPiCO especificado utilizando las Gsig, hacia el Cálculo PiCO.

```
/*Constructor de la instancia de LX, que recibe al constructor LX en Gsig,
posteriormente, se encarga de extraer el código de cada constructor X1 por
medio de separador1, y lo almacena en x1_Gsig.*/

public LX (Type lx_Gsig)
{x1_Gsig=separador1(lx_Gsig); … ; xn_Gsig = separadorn(lx_Gsig);}

//Método que se encarga de traducir el constructor LX.
public Type traductor() {
//Creación del objeto x1 a partir de x1_Gsig.
x1 = new X1(x1_Gsig); … ; x1 = new X1(x1_Gsig);

/*Obtención del código del constructor LX, mediante el ordenamiento de las
traducciones de sus componentes (constructores y símbolos)*/.
lx_final=ordenación(T⟦Sd1⟧, x1.traductor(),T⟦Sc1⟧, … , T⟦Scn-1⟧, xn.traductor(),
T⟦Sd2⟧);

return x_final;
}
```

Figura 5. Implementación de Objeto para traducción del constructor *LX*, sección Métodos.

3. DEMOSTRACIÓN DE LA VALIDEZ DE LA FUNCIÓN DE TRADUCCIÓN T EN CONJUNCIÓN CON GraPiCO.

En esta sección se muestra una lista representativa de las reglas de traducción empleadas para transformar programas GraPiCO_Textual en PiCO en conjunción con la función de traducción $T[\![\ ]\!]$ para construir una demostración matemática de la *validez* del proceso de traducción. La especificación gramatical y una explicación del Cálculo GraPiCO, así como la nueva gramática LL(1) del Cálculo PiCO, se encuentran discutidos como un ejemplo de la utilización de las Gsig en [11]).

3.1 Planteamiento recursivo de la función T.

La función T para $\varepsilon[\![X]\!]$ (ver figura 6 donde se presenta gráficamente el término Expansión[7]) mostrada en la fórmula 5, se calcula sobre la 4-upla en 6.

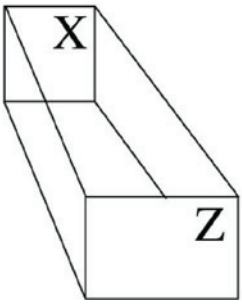

Figura 6. Expansión del constructor X en el constructor Y.

$$T[\![\varepsilon[\![X]\!]]\!] = T[\![S'_X\ Sd_{1_X}\ PF_X \sim Name(X) :\ PL_X\ Sd_{2_X}]\!] = x\_final \quad (5)$$

La Parte Lógica del constructor X consiste en $\varepsilon[\![Z]\!]$, que de ahora en adelante llamaremos Y.

$$\langle T[\![S'_X]\!], T[\![Sd_{1_X}]\!], T[\![Y]\!], T[\![Sd_{2_X}]\!]\rangle \quad (6)$$

Donde la parte lógica de X corresponde a 7.

[7] Se llamará Expansión a la acción resultante de activar un ícono para visualizar su contenido

$$Y = S'_Z\ Sd_{1_Z}\ PF_Z \sim Name(Z) :\ PL_Z\ Sd_{2_Z} \quad (7)$$

De igual forma, la función T para $\mathcal{E}[\![X]\!]$ presentada en la fórmula 8 se calcula sobre la 4-upla presentada en 9.

$$T[\![S'_Z\ Sd_{1_Z}\ PF_Z \sim Name(Z) :\ PL_Z\ Sd_{2_Z}]\!] = z\_final \quad (8)$$

$$\langle T[\![S'_Z]\!], T[\![Sd_{1_Z}]\!], T[\![PL_Z]\!], T[\![Sd_{2_Z}]\!]\rangle \quad (9)$$

Luego el planteamiento recursivo de la función T se expresa mediante el diagrama conmutativo de la figura 7:

$$\begin{array}{ccc} T[\![\mathcal{E}[\![X]\!]]\!] & \xrightarrow{sobre} & \langle T[\![S'_X]\!], T[\![Sd_{1_X}]\!], T[\![Y]\!], T[\![Sd_{2_X}]\!]\rangle\ a) \\ \downarrow & & \downarrow \\ T[\![\mathcal{E}[\![X]\!]]\!] & \xrightarrow{sobre} & \langle T[\![S'_X]\!], T[\![Sd_{1_X}]\!], z\_final, T[\![Sd_{2_X}]\!]\rangle\ b) \end{array}$$

Dado que en b) *z_final* ya está calculado y como a) y b) son del mismo tipo, podemos afirmar que b) es más sencillo que a).

Figura 7. Diagrama conmutativo de la función de traducción T

$$\{\mathcal{E}[\![X]\!] \in L(GraPiCO\_Textual)\}\ x\_final = T[\![\mathcal{E}[\![X]\!]]\!]\ \{x\_final \in L(PiCO)\}$$

Figura 8: Especificación de la función de traducción T

Dado lo anterior, verificar la *validez* de T equivale a demostrar el predicado en 10.

$$\forall_{X \in L(GraPiCO).} \mathcal{E}[\![X]\!] \in L(GraPiCO\_Textual) \Longrightarrow T[\![\mathcal{E}[\![X]\!]]\!] \in L(PiCO) \quad (10)$$

Entonces requerimos algún tipo de inducción sobre el conjunto definido en 11.

$$GraPiCO\_\mathcal{E} = \{X \in L(GraPiCO)\ :\ \mathcal{E}[\![X]\!] \in L(GraPiCO\_Textual)\} \quad (11)$$

3.2 **Relación binaria** $\succ$ en $L(GraPiCO\_textual)$

Sea la relación binaria $\succ$ en $L(GraPiCO)$, $\succ \subset L(GraPico) \times L(GraPiCO)$, definida en 13 (primero se presenta la definición del conjunto cerradura de la función de expansión en 12).

$$Ex(X)^* = \{Y : Y \in Ex(X) \vee Z \in Ex(X) \Longrightarrow Y \in Ex(Z)^*\} \qquad (12)$$

$$X \succ Y = Y \in Ex(X)^* \qquad (13)$$

Posteriormente, verificamos la transitividad de la relación $\succ$ en 14.

$$(T)\forall_{X,Y,Z \in L(GraPiCO)} \cdot \frac{\begin{matrix} X \succ Y \Longrightarrow Y \in Ex(X)^* \\ Y \succ Z \Longrightarrow Z \in Ex(Y)^* \end{matrix}}{Z \in Ex(X)^* \Longrightarrow X \succ Z} \qquad (14)$$

3.3 **Lista de reglas de traducción de GraPiCO_Textual en cálculo PiCO.**

Para mostrar la traducción de un programa GraPiCO_Textual[8] hacia PiCO, se emplea una lista de equivalencias, donde cada campo izquierdo está compuesto de un constructor de GraPiCO_Textual, y, el campo derecho se refiere a la correspondiente expresión gramatical equivalente al constructor GraPiCO Textual en términos del cálculo PiCO. A continuación un subconjunto representativo de estas reglas (mostrado en perspectiva en la figura 9 y contextualizado más adelante en la sección 3.4):

1. Traducción de un programa:

$$\begin{aligned} T[\![Programa]\!] &\equiv T[\![\{ParteFísica \sim Etiqueta : ParteLógicaPrograma\}]\!] \\ &\equiv \underbrace{T[\![\lambda]\!]}_{S'} \underbrace{T[\![\{]\!]}_{Sd_1} T[\![ParteLógicaPrograma]\!] \underbrace{T[\![\}]\!]}_{Sd_2} \end{aligned} \qquad (15)$$

- Traducción de los símbolos de sincronización:

$$T[\![\{]\!] \equiv \lambda,\ T[\![\}]\!] \equiv .$$

$$T[\![Programa]\!] \equiv T[\![ParteLógicaPrograma]\!].$$

[8] Especificación textual de un programa visual GraPiCO siguiendo las Gsig.

2. Traducción de casos de la parte lógica de un programa (lista de procesos concurrentes): a) Parte lógica de un programa compuesta de varios procesos, b) Parte lógica de un programa compuesta por un único proceso.

$$
\begin{aligned}
a) \quad & T[\![ParteLógicaPrograma]\!] \\
\equiv\ & T[\![(Proceso_1 | \cdots | Proceso_n)]\!] \\
\equiv\ & \underbrace{T[\![\lambda]\!]}_{S'}\ \underbrace{T[\![(]\!]}_{Sd_1}\ T[\![Proceso_1]\!]\ \underbrace{T[\![|]\!]}_{Sc_1} \cdots \underbrace{T[\![|]\!]}_{Sc_{n-1}}\ T[\![Proceso_n]\!]\ \underbrace{T[\![)]\!]}_{Sd_2} \\
\equiv\ & (T[\![Proceso_1]\!] | \cdots | T[\![Proceso_n]\!])
\end{aligned}
\tag{16}
$$

$$
b) \quad T[\![ParteLógicaPrograma]\!] \equiv T[\![Proceso]\!]
$$

3. Traducción de un proceso:

$$
\begin{aligned}
T[\![Proceso]\!] \ &\equiv\ T[\![Clonación\ ParteFísica \sim Etiqueta : ParteLógicaProceso]\!] \\
&\equiv\ \underbrace{T[\![Clonación]\!]}_{S'}\ \underbrace{T[\![\lambda]\!]}_{Sd_1}\ T[\![ParteLógicaProceso]\!]\ \underbrace{T[\![\lambda]\!]}_{Sd_2}
\end{aligned}
\tag{17}
$$

- Traducción de la condición replicación:

$$
T[\![Clonación]\!] \begin{cases} \boldsymbol{clone} & \text{Si } Clonación = *, \\ \lambda & \text{Si } Clonación = \lambda. \end{cases} \tag{18}
$$

4. Traducción de la parte lógica de un proceso:

Caso particular cuando la parte lógica de un proceso corresponde a un ámbito:

- Definición de ámbito:

$$
\begin{aligned}
& T[\![ParteLógicaProceso]\!] \\
\equiv\ & T[\![[ListaVariables; ListaNombres; Programa]]\!] \\
\equiv\ & \underbrace{T[\![\lambda]\!]}_{S'}\ \underbrace{T[\![[]\!]}_{Sd_1}\ T[\![ListaVariables]\!]\ \underbrace{T[\![;]\!]}_{Sc_1}\ T[\![ListaNombres]\!]\ \underbrace{T[\![;]\!]}_{Sc_2}\ T[\![Programa]\!]\ \underbrace{T[\![]]\!]}_{Sd_2}
\end{aligned}
\tag{19}
$$

- raducción de los símbolos de sincronización:

$$
T[\![[]\!] \equiv \boldsymbol{local}, T[\![]]\!] \equiv \lambda, \underbrace{T[\![;]\!]}_{Sc_2} \equiv \boldsymbol{in}
$$

$$
\underbrace{T[\![;]\!]}_{Sc_1} = \begin{cases} , & \text{Si } G_v \neq \emptyset \wedge G_n \neq \emptyset, \\ \lambda & \text{En otro caso.} \end{cases} \tag{20}
$$

Siendo G_v el Grupo de variables y G_n el grupo de nombres.

- Traducción de lista de identificadores (variables, parámetros o nombres):

$$\begin{aligned} & T[\![ListaIdentificadores]\!] \\ \equiv\ & T[\![Identificador_1, \cdots, Identificador_n]\!] \\ \equiv\ & \underbrace{T[\![\lambda]\!]}_{S'}\ \underbrace{T[\![\lambda]\!]}_{Sd_1}\ T[\![Identificador_1]\!]\ \underbrace{T[\![,]\!]}_{Sc_1} \cdots \underbrace{T[\![,]\!]}_{Sc_{n-1}}\ T[\![Identificador_n]\!]\ \underbrace{T[\![\lambda]\!]}_{Sd_2} \\ \equiv\ & T[\![Identificador_1]\!], \cdots, T[\![Identificador_n]\!] \end{aligned} \quad (21)$$

  - Traducción de un identificador (variable, parámetro, nombre o literal de recepción o delegación):

$$\begin{aligned} & T[\![Identificador]\!] \\ \equiv\ & \underbrace{T[\![\lambda]\!]}_{S'}\ \underbrace{T[\![\lambda]\!]}_{Sd_1}\ T[\![ParteFísica \sim NomIdentificador]\!]\ \underbrace{T[\![\lambda]\!]}_{Sd_2} \\ \equiv\ & NomIdentificador \end{aligned} \quad (22)$$

3.4 **El preorden** $(L(GraPiCO), \succ)$

Dibujando la relación $\succ$ obtenemos el grafo de la figura 9. El cual, en efecto, constituye el Grafo Acíclico Dirigido (desde ahora GDA) para la gramática del cálculo GraPiCO. Con relación a la sección anterior, el conjunto de reglas de traducción presentadas corresponden a la cadena de constructores resaltada en el GDA y sobre la cual se efectúa la demostración. Se tomó este grupo bien ordenado porque conserva las propiedades matemáticas pertinentes y permite la extensión de la demostración al resto de los constructores con generalización.

Consideraciones al preorden

1. Podemos observar, por las reglas de traducción presentadas en la sección 3.3 y por el GDA mostrado en la figura 9, las siguientes deducciones:

   - Para todos los constructores simples tenemos:

     - La especificación gramatical del constructor simple X es de la forma:

$$\mathcal{E}[\![X]\!] = S'_X\ Sd_{1_X}\ ParteFísica_X \sim Name(X) : Y\ Sd_{2_X}$$

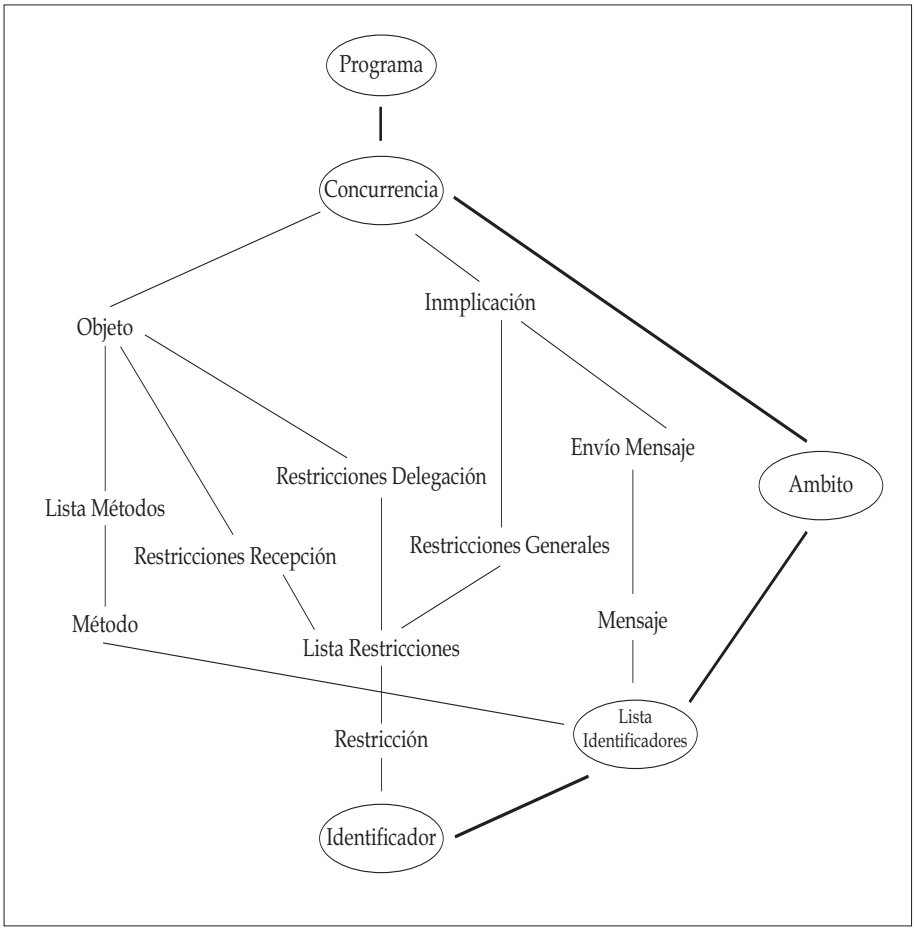

Figura 9. GDA para la Gramática de GraPiCO con cadena resaltada.

- X está compuesto por el constructor Y, Y menor que X según la relación $\succ$

$$\forall_{Y \in \mathcal{E}[\![X]\!]}. Y \in Ex(X) \Longrightarrow Y \in Ex(X)^* \Longrightarrow X \succ Y$$

- Para todos los constructores compuestos tenemos:

  - La especificación gramatical del constructor compuesto LX es de la forma:

$$\mathcal{E}[\![LX]\!] = Sd_1\ X_1\ Sc_1 \quad Sc_{n-1}\ X_n\ Sd_2$$

- El constructor compuesto LX está conformado por los constructores $X_1, \cdots, X_n$; todo X_i menor que LX según la relación $\succ$

$$\forall_{X_i \in \mathcal{E}[\![LX]\!]}. X_i \in Ex(X) \Longrightarrow X_i \in Ex(X)^* \Longrightarrow X \succ X_i$$

2. A través del GDA podemos encontrar un árbol sintáctico (AS en adelante) para cada programa GraPiCO P_i. Lo anterior, es expresado simbólicamente en 23.

$$\forall_{P_i \in L(GraPiCO)}. \exists_{AS_i} \quad (23)$$

3. Puesto que todo programa GraPiCO Pi es finito, su respectivo ASi será finito también.

4. Por 1,2 y 3 podemos observar que en un programa GraPiCO no existen sucesiones de constructores infinitas de la forma $\{X_k\}_{k \in \mathbb{N}}$ tales que $\forall_{k \in \mathbb{N}}. X_i \succ X_{i+1}$

5. Y, ya que por la ecuación 14 la relación $\succ$ sobre es $L(GraPiCO)$ es transitiva y por 4 no existen sucesiones infinitas estrictamente decrecientes, tenemos el preorden bien fundado (desde ahora *pbf*) $(L(GraPiCO), \succ)$, y podemos emplear el principio de inducción noetheriana en 10 a través de la fórmula 24.

$$\frac{\begin{array}{l}\forall_{X \in L(GraPiCO)}. \quad \{ \\ \quad \forall_{Y \in L(GraPiCO)}. X \succ Y \Longrightarrow \\ \quad \mathcal{E}[\![Y]\!] \in L(GraPiCO\_Textual) \Longrightarrow T\left[\!\left[\mathcal{E}[\![Y]\!]\right]\!\right] \in L(GraPiCO) \\ \quad \} \Longrightarrow \\ \quad \mathcal{E}[\![Y]\!] \in L(GraPiCO\_Textual) \Longrightarrow T\left[\!\left[\mathcal{E}[\![Y]\!]\right]\!\right] \in L(PiCO)\end{array}}{\forall_{X \in L(GraPiCO)}. \mathcal{E}[\![X]\!] \in L(GraPiCO\_Textual) \Longrightarrow T\left[\!\left[\mathcal{E}[\![X]\!]\right]\!\right] \in L(PiCO)} \quad (24)$$

Empleando la ecuación 11 en 24 obtenemos en 25 una expresión más concreta para aplicar la inducción noetheriana.

$$\forall_{X \in L(GraPiCO_{\mathcal{E}})}. T\left[\!\left[\mathcal{E}[\![X]\!]\right]\!\right] \in L(PiCO) \quad (25)$$

En la figura 9 podemos observar que *Identificador* es el único elemento minimal (de hecho es el mínimo) en el GDA de GraPiCO.

$$m \text{ es mínimo en } L(GraPiCO) \stackrel{\text{def}}{=} \exists!_m. \neg\exists_x. m \succ x \quad (26)$$

Con todo lo anterior, podemos aplicar el principio de inducción noetheriana de la siguiente forma:

- **Base**: De mostramos para todos los elementos minimales (en este caso solo *identificador*) que su traducción es un constructor PiCO.

$$T[\![Id\_GraPiCO]\!] \equiv T[\![ParteFísica \sim Id\_GraPiCO]\!] \equiv Identificador\_PiCO$$

- **Hipótesis**: Dado un X no minimal (para el caso debe ser diferente de *identificador*, el único minimal) suponemos que la traducción de todos los constructores de los que esté compuesto X son constructores PiCO.

$$\forall_{X \in L(GraPiCO_{\mathcal{E}}) \setminus Id}. \forall_{Y \prec X}. T[\![Y]\!] \in L(ConstructorPiCO_Y)$$

- **Inducción**: Bajo la **Hipótesis**, demostramos que la traducción de la especificación de todo constructor GraPiCO es un código PiCO.

$$\forall_{X \in L(GraPICO\_\mathcal{E})}. T[\![\mathcal{E}[\![X]\!]]\!] \in L(PiCO)$$

Tenemos que los constructores GraPiCO_Textual son de la siguiente forma:

$$\mathcal{E}[\![X]\!] = S'_X \; Sd1_X PF_X \sim Name(X) : Y \; Sd2_X$$

De la **Hipótesis**, la respectiva traducción de la especificación de X es de la próxima manera:

$$T[\![\mathcal{E}[\![X]\!]]\!] = T[\![S']\!] \; T[\![Sd_1]\!] \; ConstructorPiCO\_Y \; T[\![Sd_2]\!]$$

Resumiendo las reglas de traducción (presentando únicamente los resultados de la función *ordenación*, mediante la utilización de las funciones presentadas y la **Hipótesis**) obtenemos la siguiente lista de constructores.

1. Programa: *ProgramaPiCO.*

2. Lista de procesos: $(ProcesoPiCO_1 | \cdots | ProcesoPiCO_n)$

3. Proceso: *CondiciónReplicaciónPiCO ParteLógicaProcesoPiCO*

4. Tipos de procesos:

   - Definición de ámbito: ***local*** *ListaIdentificadoresPiCO* ***in*** *ProgramaPiCO*

     - Lista de identificadores: $IdentificadorPiCO_1, \cdots, IdentificadorPiCO\_n$

Podemos observar que todos los resultados, en efecto, son constructores PiCO y cumplen su estructura sintáctica, terminando con esto la demostración.

4. TRADUCTOR DE GraPiCO_Textual HACIA CÁLCULO PiCO

Aplicando el mecanismo introducido se realizó un proyecto[14] en lenguaje Java que tiene como objetivo diseñar e implementar un mecanismo de traducción que tome la especificación textual de un programa, construido a través del cálculo visual GraPiCO, para obtener una representación en cálculo textual PiCO e implementar una etapa de análisis semántico.

4.1 **Arquitectura del programa traductor**

Teniendo en cuenta que además de la traducción realizada sobre el código GraPiCO_Textual, se está realizando un análisis semántico al código PiCO resultante, se modela el traductor como un proceso de dos etapas principales que constituyen el núcleo del desarrollo, el traductor y el analizador semántico.

4.1.1. ***Etapa de traducción***: Dado un programa en GraPiCO_Textual, se identifica su composición, y cada una de sus partes (constructores) será tratada de manera independiente por medio de un objeto, en otras palabras, se diseñó un objeto por cada constructor de GraPiCO_Textual. Cada objeto fue implementado como una clase, las cuales tienen dos métodos principales:

- **Constructor:** Tiene el mismo nombre del constructor GraPiCO. Recibe la expresión en GraPiCO Textual y separa sus componentes, omitiendo las características visuales.

- **T**raductor: Lleva a cabo la traducción al Cálculo PiCO de una regla de traducción determinada, utilizando los componentes identificados en el constructor. (ver diagrama de la figura 10).

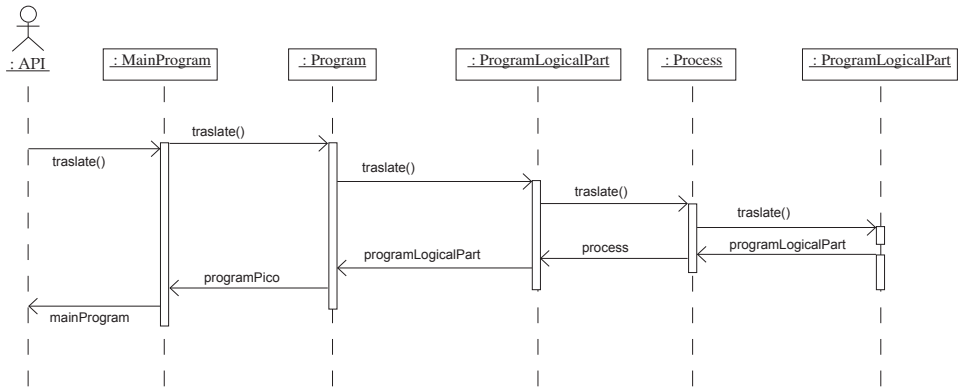

Figura 10. Diagrama de secuencia principal del traductor GraPiCO a PiCO.

4.1.2. ***Etapa de análisis semántico***: El código PiCO generado es analizado con el fin de determinar en que momento éste carece de sentido en su estructura, generando mensajes de error o advertencias semánticas. Las acciones semánticas a analizar, se obtienen cuando se traducen elementos de la estructura PiCO. La clase SemanticAnalizer provee métodos que son usados a lo largo del proceso de traducción del código GraPiCO Textual. Los mensajes que se van generando en el proceso, se almacenan en una tabla en memoria. El analizador semántico se basa en el llenado y análisis de tablas Hash.

5. CONCLUSIONES

La T_Gsig es eficiente y es una nueva forma de traducción más ágil porque utiliza el mecanismo de "scanning" y no de "parsing" para la traducción, como es usual. De otro lado, dado que la presentación práctica de T_Gsig se realizó a través de plantillas de clases, se pudo observar que es aplicable y general.

Desde el punto de vista aplicativo, con el empleo de la T_Gsig se consigue que la navegación virtual del árbol sintáctico mediante la jerarquía de

clases, permite la implementación de las fases posteriores del proceso de compilación por medio de la adición de métodos; además, la utilidad de las T_Gsig fue presentada mediante un caso de estudio real, el Cálculo Visual GraPiCO. Y desde el punto de vista teórico, gracias a la presentación y demostración de las propiedades de la T_Gsig con métodos formales, podemos concluir que es una forma de traducción sólida y válida. De la misma forma, ya que las T_Gsig extraen la información visual conservando la textual y también la estructura inicial del programa, encontramos que si existe una Correspondencia Uno a Uno entre las características de los constructores visuales y textuales del lenguaje tratado, el mecanismo de traducción preserva la semántica.

Referencias

[1] G. Álvarez, J.F Díaz, L. Quesada, F. Valencia, G. Tamura, C. Rueda and G. Assayag, "Integrating Constraints and concurrent objects in Musical applications: A calculus and its visual languaje", *Constraints Journal*¸ vol. 6, no. 1, pp.21-52, 2001.

[2] C.A. Tavera y J.F. Díaz, "Nuevo Cálculo Visual GraPiCO: Presentación de sus características fundamentales", Presentado en el 2° Congreso Colombiano de Computación - II CCC, Bogotá, Abril 2007.

[3] A. Fernández, A. Navarro, B. Fernández y J.L. Sierra, "Lenguajes de Programación, lenguajes de marcado y modelos hipermedia: Una visión interesada de la evolución de los lenguajes informáticos", en *Nuevos Géneros Discursivos: Los Textos Electrónicos*, C. López-Alonso y A. Séré, Eds. Madrid: Biblioteca Nueva, 2003, pp. 95-113.

[4] D. Raggett, "Getting started with HTML", *World Wide Web Consortion*, May 24 2005. Available: http://w3.org/MarkUp/Guide.

[5] Word Wide Web Consortium, "OWL Web Ontology Language Guide 10 Febrary 2004" in *W3C Recommendation,* M.K. Smith, C. Welty and D. McGuinness Eds. Available: http://www.w3.org/TR/owl-guide.

[6] Word Wide Web Consortium, "eXtensible Markup Language (XML) 1.0", 4th ed. 29 September 2006 in *W3C Recommendation*, T. Bray, J. Paoli, C.M. Sperberg-McQueen E. Maler y F. Yergeau Eds., Available: http://www.w3.org/TR/2006/REC-xml-20060816.

[7] H. Boley, "The Rule Markup Languaje: RDF-XML Data Model, XML schema Hierarchy, and XSL Transformations", presented at International Conference on Applications of Prolog - INAP, Tokyo, October 2001.

[8] D.Z. Hirtle, "TRANSLATOR: a TRANSlator from LAnguage TO Rules", presented at Canadian Symposium on Text Analysis -CaSTA, Fredericton, October 2006.

[9] M. Erwig and S. Kollmansberger, "Visual Specifications of Correct Spreadsheets", presented at *IEEE Symp. on Visual Languages and Human-Centric Computing*, Dallas, TX, USA, September 2005.

[10] G. Costagliola and G. Polese "Extended positional grammars", Technical Report, Dipatimento di Matematica ed Informatica. University of Salerno, Salerno, Italy, 2001.

[11] C.A. Tavera y J.F. Díaz, "Alternativa de especificación sintáctica de lenguajes visuales: Gramática de sistemas de íconos generalizados-Gsig", Presentado en el XII Congreso Argentino en Ciencias de la Computación - CACiC, San Luis, Octubre 2006.

[12] S.K. Chang, *Principles of Visual Programming Systems*. Upper Saddle River, NJ (USA): Prentice Hall, 1990.

[13] A. W. Appel, *Modern Compiler Implementation in Java*, 2nd ed. New York: Cambridge University Press, 2007.

[14] J. Noguera y A. Bermúdez, "Diseño e implementación de la traducción desde la especificación textual del Cálculo Visual GraPiCO hacia una representación semánticamente válida en Cálculo PiCO", Proyecto de grado, Escuela de Ingeniería de Sistemas y Computación, Universidad del Valle, Septiembre 2005.