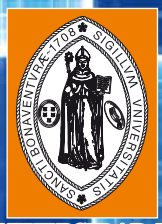
Diseño, implementación y corrección de GraPiCO

Un cálculo visual, orientado al objeto y por restricciones compilado a PiCO



UNIVERSIDAD DE SAN BUENAVENTURA SECCIONAL CALI

CARLOS ANDRÉS TAVERA ROMERO

Diseño, implementación y corrección de GraPiCO
Un cálculo visual orientado al objeto y por restricciones compilado a PiCO

Universidad de San Buenaventura, seccional Cali
Editorial Bonaventuriana

Título: ***Diseño, implementación y corrección de GraPiCO***
Un cálculo visual orientado al objeto
y por restricciones compilado a PiCO

Autor: Carlos Andrés Tavera Romero (catavera@usbcali.edu.co)

ISBN: 978-958-8436-35-7

Rector
Fray Álvaro Cepeda van Houten, OFM

Secretario
Fray Hernando Arias Rodríguez, OFM

Vicerrector Académico
Juan Carlos Flórez Buriticá

Vicerrector Administrativo y Financiero
Félix Remigio Rodríguez Ballesteros

Directora Investigaciones
Angela Rocío Orozco Zárate
e-mail: arorozco@usbcali.edu.co

Director Proyección Social
Ricardo Antonio Bastidas

Coordinador Editorial Bonaventuriana
Claudio Valencia Estrada
e-mail: clave@usbcali.edu.co

Diseño de carátula: Edward Carvajal A.

La Umbría, carretera a Pance
A.A. 25162
PBX: (572)318 22 00 – (572)488 22 22
Fax: (572)488 22 31/92
www.usbcali.edu.co • e-mail: EditorialBonaventuriana@usbcali.edu.co
Cali - Colombia, Sur América

Cali, Colombia
Diciembre de 2010

Este libro por sus especificidades técnicas fue construido en el programa *LaTeX*.

Universidad de
San Buenaventura
seccional cali

Diseño, implementación y corrección de GraPiCO

Un cálculo visual orientado al objeto y por restricciones compilado a PiCO

Carlos Andrés Tavera Romero

2010

A Dios que todo lo rige,
a mi madre por su inconmensurable amor,
a Yenny por ser mi α y ω,
y a la Universidad de San Buenaventura, seccional Cali, por apoyar decisivamente la investigación.

Agradecimientos

Quiero expresar mis agradecimientos a las directivas y colaboradores de la Universidad de San Buenaventura, seccional Cali, que ayudaron en la conclusión de este proyecto:

– Al Rector Fray Álvaro Cepeda van Houten, OFM, por su liderazgo.

– Al Vicerrector Académico, Juan Carlos Flórez Buriticá, por sus directrices.

– Al decano de la Facultad de Ingeniería, ingeniero Claudio Camilo González Clavijo, por la correcta gerencia de los procesos investigativos.

– A la directora de Investigaciones, la psicóloga Ángela Rocío Orozco; a su asistente, Karen Nahyr Sierra Ortiz y a su secretaria Yamilé Quinayás, por su constante apoyo.

– Al director de la Editorial Bonaventuriana, abogado Claudio Valencia Estrada, y a su equipo de trabajo por el apoyo ofrecido en todos los aspectos relacionados con la revisión final, el diseño y publicación de nuestros productos y/o resultados de investigación.

– Al coordinador de investigaciones de la Facultad de Ingeniería, ingeniero Luis Merchán Paredes, por apoyar los procesos investigativos del grupo de investigación del LIDIS.

– A mis compañeros del LIDIS, ingenieros Yenny Alexandra Méndez Alegría, Christian Gustavo Arias Iragorri, Diego Armando Gómez Mosquera y Carlos Andrés Mera Banguero, por su incansable aporte de fuerza y aliento.

– A mis amigos y estudiantes de la Universidad de San Buenaventura, seccional Cali, que me ayudan a evolucionar como persona, investigador y docente.

De igual manera, agradezco a otras personas por su cooperación en el desarrollo de la investigación y el informe final.

– A mi mentor, el profesor Juan Francisco Díaz Frías, de la Universidad del Valle, por su guía, ayuda, apoyo y valiosos consejos.

– A la comunicadora social Yenny Viviana Cruz Perez, profesora de la Universidad Autónoma de Occidente, por la edición de estilo del material escrito.

Índice general

I Preámbulo **14**

II Desarrollo **28**

1. Nueva gramática del cálculo PiCO **29**
1.1. Gramática del cálculo PiCO en forma LL(1) 29

2. El nuevo cálculo visual GraPiCO, sus bases formales **33**
2.1. Alfabeto del cálculo visual GraPiCO 34
2.2. Sintaxis del cálculo visual GraPiCO 35
2.3. Congruencia estructural y relación de equivalencia del cálculo visual GraPiCO 38

3. Especificación textual y sintaxis visual de GraPiCO **41**
3.1. Pertinencia de una especificación textual de GraPiCO 41
3.2. Breve panorama de la especificación textual de los VPL 42
3.3. Especificación sintáctica de lenguajes visuales 44
3.3.1. Gramáticas posicionales extendidas 44
3.4. Alternativa de especificación sintáctica: gramática de sistemas de icónicos generalizados - Gsig 47
3.4.1. Composición de los sistemas icónicos 48
3.4.2. Casos de especificación de constructores visuales 48
3.4.3. Definición del proceso de construcción de las Gsig a partir de la especificación de los constructores visuales 49
3.4.4. Ejemplo de utilización de las Gsig 51
3.5. Lista de fórmulas de especificación de GraPiCO hacia GraPiCO_Textual . . . 53
3.6. Gsig de GraPiCO 66
3.7. La relación de reducción de los constructores visuales GraPiCO 75
3.7.1. Relación de reducción en el cálculo PiCO 75
3.7.2. Relación de reducción para los programas GraPiCO 76

4. Reglas de traducción GraPiCO_Textual-cálculo PiCO **84**
4.1. Introducción 84
4.2. Definición del mecanismo de traducción T_Gsig 85
4.3. Prueba de la *corrección* de la función de traducción $\mathcal{T}$ 91

4.3.1. Demostración de la *validez* de la función de traducción $\mathcal{T}$ 91
4.3.2. Demostración de la *completitud* de la función de traducción $\mathcal{T}$ 104

5. Prueba de consistencia de la semántica entre los programas GraPiCO y PiCO 107

6. Comprobación sintáctica de la especificación textual de lenguajes visuales mediante símbolos de sincronización 134
6.1. Alternativa para comprobación sintáctica de lenguajes visuales 135
6.1.1. Comentarios sobre el mecanismo de análisis sintáctico planteado . . . 139
6.2. Implementación de GraPiCO 139

7. Breve discusión de las ventajas de los lenguajes visuales y el cálculo GraPiCO 141
7.1. Expresividad del cálculo visual GraPiCO 142
7.1.1. Programa PiCO: salón de clase 143
7.1.2. Programa GraPiCO: salón de clase 146
7.2. Ejemplo final 150

III Epílogo 153

8. Resultados, conclusiones y trabajo futuro 154
8.1. Resultados 154
8.1.1. Resultados directos 154
8.1.2. Resultados indirectos 154
8.2. Conclusiones 155
8.3. Trabajo futuro 157

Índice de figuras

1. Gramática de PiCO. 22
2. Etapas de la compilación en el lenguaje CORDIAL. 24
3. Vista de un programa CORDIAL. 25
4. Etapas de la compilación en el nuevo lenguaje GraPiCO. 27

1.1. Gramática LL(1) del cálculo PiCO. 30
1.2. Gramática de las restricciones en PiCO. 31
1.3. Gramática de las restricciones de recepción. 32

2.1. Ícono Ic' (representa la figura en su totalidad): expansión del ícono etiquetado con Ic en los íconos con etiquetas Ic_1, Ic_2 ... Ic_n dentro del ícono Ic' . . 33
2.2. Constructores del cálculo visual GraPiCO (Parte 1). 34
2.3. Constructores del cálculo visual GraPiCO (Parte 2). 35
2.4. Programa P y su conjunto de procesos concurrentes $P1,\ P2,\ P3,\ \cdots$. . . . 36
2.5. Proceso $P2$ con la señal de replicación. 36
2.6. Restricción $C1$ y la operación de comparación $\boldsymbol{sender} < A + V$. 37
2.7. Constructor de consulta de restricciones con antecedente A (con restricciones $C1,\ C2,\ C3,\ \cdots$) y consecuente el programa P. 37
2.8. Método M con argumentos $A1,\ \cdots, An$ y el programa P. 37
2.9. Constructor objeto con restricciones de recepción (representadas por el ícono del buzón cerrado), el objeto O y las restricciones de delegación (representadas por el buzón abierto). 38
2.10. Constructor de nuevo ámbito con conjunto de variables $V1,\ V2,\ V3, \cdots$; conjunto de nombres $M, \cdots$; y programa P. 38
2.11. Procesos equivalentes luego de cambiar las apariciones de I por I'. 39
2.12. Proceso $P1$ actuando concurrentemente con un proceso vacío $P2$ y congruente con el proceso $P1$. 39
2.13. Objetos equivalentes con métodos y restricciones equivalentes. 39
2.14. Proceso replicado P actuando concurrentemente con P y equivalente a P. . . 40
2.15. Declaración de nuevos identificadores en un proceso vacío y su equivalencia con el mismo. 40
2.16. Procesos equivalentes con diferente orden de declaración de identificadores. . 40

3.1. Conjunto de íconos con etiquetas Ic_1, Ic_2, ... , Ic_n dentro del ícono Ic'. . . . 44

3.2. Gramática posicional extendida de un grupo íconos: $Ic_1, Ic_2, \ldots, Ic_n$ dentro del ícono Ic'. 45
3.3. Ícono Ic' (representa la figura en su totalidad): expansión del ícono etiquetado con Ic en los íconos con etiquetas Ic_1, Ic_2, ... , Ic_n dentro del ícono Ic'. 45
3.4. Gramática posicional extendida del ícono Ic', compuesto de Ic y su expansión Ic' 46
3.5. a) Programa CORDIAL y b) su estructura hashtable. 47
3.6. Casos de constructores visuales modelados por las Gsig. 51
3.7. Especificación de un programa. 53
3.8. Especificación de procesos concurrentes 53
3.9. Especificación de una condición de persistencia 54
3.10. Especificación de un proceso 54
3.11. Grupo de variables $G_v = \{V_1,\ V_2,\ \cdots,\ V_n\}$ y grupo de nombres $G_n = \{N_1,\ N_2,\ \cdots,\ N_m\}$ en la creación de ámbito para grupos de variables y nombres. 54
3.12. Especificación de un grupo de variables o de un grupo de nombres en la creación de un nuevo ámbito. 55
3.13. Especificación de una creación de ámbito para variables y nombres. 55
3.14. Especificación de un método. 56
3.15. Especificación de una variable. 56
3.16. Especificación de un objeto. 56
3.17. Especificación de definición de objeto. 57
3.18. Lista de elementos (métodos) de objeto: $L_M = \{E_1,\ \cdots,\ E_m\}$. 57
3.19. Especificación de una lista de elementos de objeto. 57
3.20. Especificación de un método. 58
3.21. Grupos de argumentos $G_a = \{A_1,\ \cdots,\ A_n\}$. 58
3.22. Especificación de un grupo de argumentos en un método. 58
3.23. Especificación del cuerpo de un método. 59
3.24. Especificación de un argumento. 59
3.25. Especificación de restricciones de objeto. 59
3.26. Especificación de un antecedente en una implicación. 60
3.27. Especificación de una implicación. 60
3.28. a) Grupo de Relaciones en una aplicación: $G_r = \{R_1 = (A_1, I_1),\ R_2 = (A_2, I_2),\ \cdots,\ R_n = (A_n, I_n)\}$, b) Relación de aplicación. 61
3.29. Especificación de un grupo de Relaciones en una aplicación. 62
3.30. Especificación de una relación de aplicación. 62
3.31. Especificación de un puerto de método. 63
3.32. Especificación de un envío de mensaje. 63
3.33. Ejemplo de grupo de restricciones. 64
3.34. Especificación de un grupo de restricciones 64
3.35. Especificación de la expansión de una restricción: a) restricción compuesta por una comparación C y una operación O, b) restricción compuesta por una comparación C. 65
3.36. Especificación de los componentes de una restricción: a) Comparación C, b) Operación O. 65

3.37. Especificación de un literal receptor o delegador en una restricción de objeto. 65
3.38. Programa GraPiCO compuesto por los procesos concurrentes: P_1, P_2, P_3... . 66
3.39. Sección de gramática para programa. 67
3.40. Ejemplos de los tipos de procesos vistos "de cerca": a) Definición de ámbito, b) Objeto, c) Implicación. 67
3.41. Replicación de procesos: a) Operador icónico para la replicación, b) Proceso P_2 con replicación. 68
3.42. Proceso: definición de ámbito. Las variables: V_1, V_2, V_3... y el mensaje M, son locales al programa P. 68
3.43. Sección de gramática para el proceso definición de ámbito. 69
3.44. a) Restricciones en objeto, b) Definición de objeto. 69
3.45. Método M compuesto de la lista de argumentos: $A_1, \ldots, A_n$ y el programa P 70
3.46. Sección de gramática para el proceso objeto. 70
3.47. Tipos de restricciones en objeto: a) Restricciones para recepción, b) Restricciones para delegación. 71
3.48. Ejemplos de restricciones en objeto: a) Ejemplo de una restricción para recepción, b) Ejemplo de una restricción para delegación. 71
3.49. Sección de gramática para las restricciones de objeto. 72
3.50. Sección de gramática para las restricciones de objeto (continuación). 73
3.51. Tipos de antecedentes: a) Imposición de restricciones, b) Consulta de restricciones, c) Envío de mensaje. 73
3.52. Sección de gramática para el proceso implicación y su componente antecedente. 74
3.53. Consecuente. 75
3.54. Sección de gramática para el consecuente de una implicación 75
3.55. Sistema de transición. 76
3.56. Reducción de concurrencia. 77
3.57. Reducción de comunicación. 78
3.58. Reducción de delegación. 80
3.59. Reducción de una consulta de restricciones. 81
3.60. Reducción de imposición de restricciones. 82
3.61. Reducción de creación de ámbito. 83

4.1. Diseño de objeto del constructor simple X. 86
4.2. Implementación de objeto del constructor simple X, sección: atributos. . . . 86
4.3. Implementación de objeto del constructor simple X, sección: métodos. 87
4.4. Diseño de objeto del constructor visual compuesto L_X. 89
4.5. Implementación de objeto para la traducción del constructor visual compuesto L_X. 89
4.6. Implementación de objeto para traducción del constructor visual compuesto L_X. 90
4.7. Expansión del constructor X en el constructor Z. 91
4.8. Diagrama conmutativo del planteamiento recursivo de la función de traducción $\mathcal{T}$. 92
4.9. Especificación de la función traducción $\mathcal{T}$. 92
4.10. Grafo de la relación $\succ$ en ℓ(GraPiCO). 100
4.11. Constructor visual compuesto L_X y sus constructores X_1, X_2, $\cdots$ y X_n. . . 101

5.1. Diagrama conmutativo de la corrección de las reglas de traducción. 107
5.2. Premisas para el proceso de traducción. 108
5.3. Resolución de la traducción de la especificación del operador visual de creación de nuevo ámbito. 109
5.4. Resolución de la traducción de la especificación del constructor visual de programa. 110
5.5. Diagrama conmutativo de la corrección de las reglas de traducción del constructor visual de creación de ámbito. 110
5.6. Premisas para el proceso de traducción. 111
5.7. Resolución de la traducción de la especificación del constructor visual de concurrencia 1. 112
5.8. Resolución de la traducción de la especificación del constructor visual de concurrencia 2. 113
5.9. Diagrama conmutativo de la corrección de las reglas de traducción del constructor de concurrencia. 113
5.10. Premisas para el proceso de traducción. 114
5.11. Resolución de la traducción de la especificación de imposición de restricciones. 115
5.12. Resolución de la traducción de la especificación del constructor visual de programa. 116
5.13. Diagrama conmutativo de la corrección de las reglas de traducción del constructor visual de imposición de restricciones. 116
5.14. Premisas para el proceso de traducción. 117
5.15. Resolución de la traducción de la especificación del constructor visual de consulta de restricciones 1. 118
5.16. Resolución de la traducción de la especificación del constructor visual de programa. 119
5.17. Diagrama conmutativo de la corrección de las reglas de traducción del constructor visual de consulta de restricciones. 119
5.18. Premisas para el proceso de traducción. 120
5.19. Resolución de la traducción de la especificación del constructor comunicación. 121
5.20. Resolución de la traducción de la especificación del constructor comunicación. (continuación a) 122
5.21. Resolución de la traducción de la especificación del constructor comunicación. (continuación b) 123
5.22. Resolución de la traducción de la especificación del constructor comunicación. (continuación c) 124
5.23. Resolución de la traducción de la especificación del constructor comunicación. (continuación d) 125
5.24. Resolución de la traducción de la especificación del constructor comunicación. (continuación e) 126
5.25. Resolución de la traducción de la especificación del constructor comunicación. (continuación f) 127
5.26. Resolución de la traducción de procesos concurrentes en una comunicación. . 128
5.27. Resolución de la traducción de la especificación de procesos concurrentes resultantes de una comunicación (continuación). 129
5.28. Diagrama de la corrección de las reglas de traducción de comunicación. . . . 129

5.29. Premisas para el proceso de traducción. 130
5.30. Resolución de la traducción de la especificación del constructor delegación. . 131
5.31. Resolución de la traducción de la especificación de la creación del nuevo ámbito para la variable J. 132
5.32. Diagrama conmutativo de la corrección de las reglas de traducción del constructor visual de delegación. 133

6.1. Compilador escalonado. 134
6.2. Ventana de constructores activos y ventana de expansión activa. 137
6.3. Íconos permitidos en la ventana de expansión activa del ícono del constructor X. 138

7.1. Programa salón de clase (definición de alumno “egoísta”). 144
7.2. Programa salón de clase (definición de alumno “generoso”). 145
7.3. Programa salón de clase (definiciones de profesor, y verificación y petición de esfero). 146
7.4. Programa salón de clase GraPiCO (nivel de abstracción 1). 147
7.5. Programa salón de clase GraPiCO (nivel de abstracción 2). 148
7.6. Programa salón de clase GraPiCO (más niveles de abstracción). 149
7.7. Ventana inicial del programa GraPiCO que calcula factorial (Nivel 1). 150
7.8. Ventana resultado de la expansión de los íconos presentes en la ventana inicial. 151
7.9. Ventana que muestra la expansión de los métodos de los objetos de la ventana inicial. 152

PARTE I

PREÁMBULO

El presente documento muestra los resultados de la investigación realizada para el desarrollo del cálculo visual GraPiCO.

El informe tratará el siguiente orden de ideas:

1. Preámbulo: Resumen y un grupo de introducciones a los contenidos que se requieren para adquirir una contextualización del tema discutido.
2. Desarrollo: Planteamiento del problema encontrado y una propuesta de solución.
3. Epílogo: Presentación de las conclusiones, resultados y trabajo futuro.

Palabras clave

lenguajes de programación visual, cálculos computacionales, compiladores y demostración de corrección.

Resumen

En el presente documento se propone un nuevo cálculo visual como solución ante el requerimiento del empleo de multimedios para ampliar los horizontes en cuanto a la difusión y empleo del cálculo textual PiCO dadas las más recientes tendencias y tecnologías.

El desarrollo de la investigación se presentará con la disposición siguiente:

- Alfabeto y sintaxis del cálculo visual GraPiCO: se muestra el conjunto de símbolos empleados en el lenguaje y su localización en un programa. Estos aportes fueron publicados en [TD07].
- Especificación textual y sintáxis visual: donde se pone en consideración una novedosa forma de describir textualmente la sintaxis de un lenguaje visual como el cálculo GraPiCO. El aporte más importante de la investigación, fue presentado en [TD06].
- Mecanismo de traducción: en esta parte del documento se presenta una forma de transformar la representación textual de un programa GraPiCO en código intermedio. Uno de los aportes fundamentales, difundido en [TD09].
- Demostración de la consistencia de la semántica: en este apartado se comprueba que la función de especificación textual y las reglas de traducción preservan el significado de los programas.
- Comprobación sintáctica de la especificación textual de lenguajes visuales mediante símbolos de sincronización: capítulo donde se propone una nueva forma de analizar sintácticamente los lenguajes visuales por medio de los símbolos de sincronización generados en sus correspondientes representaciones gramaticales. Esta contribución se expuso en [TDS$^+$07].
- Breve discusión de las ventajas de los lenguajes visuales: aquí se discuten las ventajas de los lenguajes visuales frente a los textuales. Este análisis se puede encontrar en [TD08].

Introducción

Bisonte de Altamira

Toro de Lascaux

Lo visual

Los humanos al enfrentar un problema primero imaginan la solución y luego, a partir de estos dibujos creados con el pensamiento, construyen los resultados: cocinan alimentos, diseñan vehículos, escriben poemas y componen música. Lo visual es inherente en las personas, por esta razón las primeras manifestaciones realmente escritas de la humanidad[1] se presentaron 14 siglos después que las gráficas[2] y que l@s niñ@s hacen dibujos tres años antes de sus textos iniciales[3].

En el terreno tecnológico la evolución se dio de manera inversa; primero se procesaron textos y luego imágenes. Las primeras impresoras sólo podían plasmar unas rudimentarias letras y ni pensar en dibujos; los monitores en sus orígenes únicamente presentaban pocas líneas de texto; los lenguajes de programación y los sistemas operativos iniciales ni siquiera tenían entorno gráfico y los electrodomésticos no llevaban pantallas que desplegaran información; esto sucedía porque aunque lo gráfico es innato en los seres humanos, para manipular artificialmente figuras se requieren niveles de procesamiento muy superiores. Este último es el motivo por el cual, aunque las teorías de la geometría computacional y de procesamiento de imágenes ya estaban muy avanzadas en el siglo XX, apenas ahora los procesadores son lo suficiente veloces y económicos como para poder ejecutar los sistemas operativos totalmente visuales y los lenguajes de programación realmente gráficos en sus albores.

[1] Escritura alfabética sirio-palestina 1.500 A.C.

[2] Pictografías del paleolítico superior en las cuevas de Altamira-España (época estilo III) y Lascaux-Francia (epoca magdaleniense) 15.000 A.C. www.artehistoria.com

[3] En la etapa pre-escolar (3 años) se comienza la elaboración de dibujos, en la etapa escolar (6 años) se inicia la lectura y escritura www.urbanext.uiuc.edu

Los anteriores antecedentes, sumados a las tendencias y costumbres adquiridas por el surgimiento de nueva tecnología que permite desarrollos más exigentes como los multimedios, hacen que los usuarios exijan cada vez más elementos visuales; un celular sin pantalla a colores era ya obsoleto hace unos años, ahora uno sin cámara y despliegue de vídeo es poco atractivo; para atraer a sus clientes los supermercados y servitecas invierten buen presupuesto en la instalación y divulgación de sus antenas para red inalámbrica, las que permiten públicamente acceder a Internet. Lo visual es muy atractivo para el humano y le facilita la interacción con otros elementos, pues se siente identificado y más cercano. Dentro de estas aplicaciones atractivas para el humano se encuentran los lenguajes visuales (conjunto donde están los cálculos visuales) los que a partir de íconos y gráficas hacen de la programación una experiencia más agradable; en la siguiente sección se presenta una introducción a ellos.

Los lenguajes visuales

En la modelación de algoritmos o programación los símbolos utilizados pueden tener varias formas. La primera en surgir y más difundida es la programación textual donde toda la definición se hace con un código representado en caracteres. Debido a la necesidad de programas cada vez más extensos, para la solución de problemas cada vez más complejos, se precisan mecanismos que le permitan al usuario final un mayor nivel de abstracción, facilidades para el análisis y la utilización de un lenguaje más cercano a los modelos mentales. Por esto emerge la programación visual para ofrecer una aproximación a la solución.

En cuanto a la definición básica, la programación visual se refiere al mecanismo que permite la modelación total y especificación de soluciones a problemas mediante gráficas; esto sugiere que en ninguna parte de los programas debería requerirse código textual.

Las investigaciones en el campo de los lenguajes visuales están encaminadas a desarrollar herramientas efectivas de programación que involucren, únicamente, la definición gráfica de los problemas. Esto contrasta con la metodología que se popularizó en los años noventa, ejemplificada en herramientas como Visual C++, Visual Basic, Delphi y Java, lanzadas al mercado como lenguajes visuales, pero más encaminadas a programar aplicaciones para el entorno operativo Windows que como verdaderos lenguajes gráficos, pues el usuario debe, además de entender el significado de los íconos, conocer un amplio lenguaje totalmente textual. Un análisis sobre las características de los verdaderos lenguajes visuales se encuentra en [Nad04].

Consecuente con esta evolución diversos laboratorios están investigando metodologías de procesamiento de lenguaje plenamente visual hacia algún lenguaje objeto. Ejemplos particulares de este desarrollo son los trabajos de la Universidad de Pittsburgh dirigidos por el profesor Shi-Kuo Chang[4], donde su objetivo principal es definir mecanismos de traducción dirigidos por la sintaxis relacional y el análisis de la semántica visual.

[4] www.cs.pitt.edu/~chang/

Se puede encontrar uno de los primeros pasos en la compilación de lenguajes visuales en un famoso artículo del profesor Chang [Cha88]; posteriormente, junto con el profesor Costagliola (referenciado más adelante) crea un artículo en el cual convergen la parte teórica y práctica en el análisis sintáctico [Cha99].

Otro grupo interesado en lenguajes visuales es el que investiga gramáticas y especificaciones semánticas, dirigido por el profesor Gennaro Costagliola del Dipartimento Di Matematica ed Informatica, Università di Salerno, Italy[5]; algunas de sus publicaciones aparecen en las revistas de lenguajes visuales y computación (referenciadas en los siguientes párrafos).

Finalmente, existe el grupo liderado por el profesor Martin Erwig de FernUniversitad Hagen, Hagen, Germany[6], de igual forma interesado en la formalización de los lenguajes visuales, de manera más específica, en su semántica. En sus albores presentó herramientas para la formalización de las propiedades semánticas, tomando como base la sintaxis visual abstracta [Erw97]; posteriormente introdujo la noción de grafo visual [Erw98a] y luego extendió los desarrollos en cuanto a los estudios matemáticos en la sintaxis visual abstracta [Erw98c]. Su contribución más nueva es un estudio sobre la inferencia visual de tipos [Erw06].

Tomando la perspectiva académica, muchas investigaciones recientes se han encaminado al desarrollo de formalismos para la definición y diseño de programas en ambientes visuales. En particular, ha sido introducido un buen número de formalismos gramaticales para describir y analizar de manera sintáctica los lenguajes visuales. Básicamente se pueden distinguir dos desarrollos para representar sentencias de estos lenguajes visuales: La *representación basada en relaciones* describe una sentencia como un conjunto de objetos gráficos y un conjunto de relaciones entre ellos [TVG94] y [Wit96]. Y la *representación mediante atributos* [CLO94] concibe una sentencia como un conjunto de objetos gráficos con atributos.

En la academia se han desarrollado lenguajes de programación visual de gran formalismo y expresividad como Pictorial Janus (PJ); introducido por Kahn y Saraswat en 1989 que es basado en el lenguaje de programación lógico concurrente Janus. Un programa es un dibujo y su ejecución es una animación. Para mayor información se puede consultar la página http://jerry.c-lab.de/ wolfgang/PJ/introduction.html.

Desde la industria existe el producto LabVIEW (*Laboratory Virtual Instrumentation Engineering Workbench*), una plataforma y un ambiente de desarrollo para un lenguaje de programación visual autoría de National Instruments. Este es denominado G, un poderoso lenguaje gráfico que emplea flujo de datos estructurado. En un inicio desarrollado para los computadores Apple Macintosh en 1986, LabVIEW es usado para adquisición de datos, instrumentación de control y automatización industrial sobre una variedad de plataformas entre las cuales está Microsoft Windows, varias versiones de UNIX, Linux, y Mac OS. El más reciente LabVIEW es la versión 8.0. Para más detalles se puede consultar la página http://www.ni.com/labview/. Los diagramas de flujo de datos han acompañado a los programadores desde los primeros lenguajes como ayuda educativa, ahora con sistemas de cómputo más ágiles y desarrollos, como la internet, han surgido diversas herramientas para asistir la

[5] www.dmi.unisa.it/people/costagliola
[6] www.fernuni-hagen.de/FeU/such_f.html

programación como ZenFlow [MPPJ05], JOpera [PA05] y BioOpera [PA03] (lenguajes y ambientes de programación para la gestión de servicios en la web); la modelación de un programa se hace por medio de grafos dirigidos, mecanismo muy particular aunque bastante efectivo para esta clase de lenguajes visuales.

Muchos de los formalismos propuestos soportan *analizadores pictóricos independientes del orden* que procesan los objetos en la entrada, es decir, sin tener en cuenta algún criterio de orden. Los formalismos de gramáticas pictóricas por capas [Gol91], gramáticas relacionales [CGs+91] y gramáticas por restriccion de conjuntos [Mar94] hacen parte de esta clase. En general, y en el peor caso, un analizador independiente del orden procede con metodología puramente ascendente. Para limitar el costo computacional del análisis sintáctico las clases de formalismos GPC, GR y GRC han sido definidas para proveer los correspondientes analizadores con capacidades predictivas que restringen el espacio de búsqueda. Dado esto, para mejorar la eficiencia, *analizadores pictóricos predictivos* como el pLR [CTOL97], basado en gramáticas posicionales y el presentado en [Wit92], sobre la base de en gramáticas relacionales, han sido también definidos.

Desde su introducción [CC90], Las *gramáticas posicionales* han estado en constante evolución, con el fin de representar clases de lenguajes visuales cada vez más extensos. Estas fueron concebidas como una extensión simple de las gramáticas independientes del contexto para el caso de los lenguajes simbólicos de dos dimensiones. Las sentencias generadas eran fundamentalmente matrices de símbolos con las cuales se podían describir expresiones aritméticas simples de dos dimensiones. La definición de gramáticas posicionales ha sido fuertemente influenciada por la necesidad de tener un analizador sintáctico eficiente capaz de procesar los lenguajes generados. Dada la analogía con las gramáticas de cadenas, ha sido muy natural la utilización de las técnicas LR. El resultado fue la definición del análisis sintáctico posicional pLR. Esta metodología ha evolucionado en paralelo con el formalismo gramatical hasta su más reciente extensión presentada en [CP01].

La programación por restricciones

La programación por restricciones es un paradigma emergente para la descripción declarativa y solución eficiente de una gran cantidad de problemas, en paticular los combinatorios, que están presentes en áreas de planificación y programación.

Las restricciones han emergido, de manera reciente, como una línea de investigación que combina desarrollos de un buen número de campos, incluyendo inteligencia artificial, lenguajes de programación, computación simbólica, y lógica computacional. Por otro lado, las redes de restricciones y problemas de satisfacción de restricciones han sido estudiados en inteligencia artificial desde los años setenta. La utilización sistemática de restricciones en la programación aparece ya en los aos ochenta. En la programación por restricciones el proceso de programar consiste en la generación de requerimientos (restricciones) y la solución de estos mismos, mediante resolutores (*solvers*) especializados.

Los primeros lenguajes de programación por restricciones fueron propuestos por Steele [Ste80] y Borning [Bor81]. Estos permitían al usuario describir el comportamiento del sistema utilizando conjuntos estáticos de restricciones. En el lenguaje de Steele, únicamente se permitía la definición de restricciones por medio de macros simples, mientras que en el sistema Thinglab de Borning se construían a través de una interfaz gráfica.

En esencia estos dos lenguajes se orientaron hacia la modelación de dominios, pero no emplearon el poder computacional de la verdadera programación por restricciones, ya que eran los primeros desarrollos prácticos y la teoría para su implementación estaba muy incipiente.

Los lenguajes de programación lógicos por restricciones nacieron en tres sitios diferentes. En Melbourne, Joxan Jaffar y Jean Louis Lassez ofrecieron sus fundamentos semánticos [JL87]. Luego con otros investigadores desarrollaron el lenguaje CLP(R). Entre tanto, en Marsella, Alain Colmerauer y su grupo desarrollaron una extensión de Prolog llamada Prolog-III [Col90], y en Munich, en el Centro Europeo de Investigación en Computación Industrial, un grupo liderado por Mehmet Dincbas desarrolló el lenguaje CHIP [Din88], que luego fue empleado en la solución de diversos tipos de problemas combinatorios.

En lo que respecta a los lenguajes por restricciones concurrentes, se originaron desde el trabajo en los lenguajes lógicos concurrentes. Maher y Saraswat [Sar93] generalizaron las ideas anteriores hacia el contexto de las restricciones; Kanellakis introdujo el paradigma de consulta por restricciones [KKR95], mientras que Darlington fue el primero en estudiar la forma funcional por restricciones [DYP92].

A su vez, un grupo de desarrollo encabezado por el profesor Smolka creó un lenguaje de programación llamado Oz [Smo95], cuya novedad consistió en la extensión del paradigma de programación concurrente por restricciones mediante la inclusión del empleo de funciones. Despues, se presentó el lenguaje Mozart[7], una evolución multiparadigma de Oz.

Otro lenguaje, Kaleidoscope [BFB90], desarrollado por Borning y Freeman-Benson con ayuda de otros investigadores, fue el primer lenguaje imperativo por restricciones.

De los primeros lenguajes por restricciones orientados al objeto fue un Lisp orientado al objeto llamado PECOS, que fue el precursor del lenguaje Ilog Solver (basado en C++), que se puede conseguir comercialmente.

Simbiosis entre la programación visual y la programación concurrente por restricciones

En el 1996 el grupo AVISPA[8] se interesó por la investigación en la solución de problemas que involucran el empleo de restricciones e inició el desarrollo de un cálculo para modelar

[7] www.mozart-oz.org

[8] Ambientes VISuales de Programación aplicativa, http://avispa.puj.edu.co

las características que debería tener un lenguaje de programación por restricciones moderno y novedoso, que luego se puediera utilizar como base para una herramienta de programación que asistiera la composición musical; este cálculo se bautizó PiCO (*π calculus and concurrent objects*) [ADQ$^+$98]. El cálculo propuesto abarcó la especificación mediante objetos, la imposición de restricciones, y ofreció todas las características requeridas para considerarlo un cálculo para programación de restricciones orientado al objeto. Como los objetos actúan de manera concurrente, por esto se llamó cálculo de restricciones y objetos concurrentes [QR98]. PiCO experimentó posteriormente una evolución, pero conservó su nombre [ADQ$^+$01] (la gramática de PiCO final es presentada en la Figura 1). Por ello, en adelante PiCO se referirá a la última versión y PiCO^{-1} a la original (PiCO resulta ser un superconjunto de PiCO^{-1}).

Proceso Normal:			
N	$\rightarrow$	O	Inactividad o proceso nulo
	$\mid$	$I \triangleleft m$ **then** P	mensaje enviado por I
	$\mid$	$(\phi_{sender}, \delta_{forward}) \triangleright M$	Objeto con condición de delegación $\delta_{\mathbf{forward}}$ guardado por la restricción ϕ
Proceso restricción:			
R	$\rightarrow$	**tell** ϕ **then** P	Proceso **tell**
	$\mid$	**ask** ϕ **then** P	Proceso **ask**
Procesos:			
P, Q	$\rightarrow$	**local** x **in** P	Nuevas variables x en P
	$\mid$	**local** a **in** P	Nuevo nombre a en P
	$\mid$	N	Proceso Normal
	$\mid$	$P \mid Q$	Composición
	$\mid$	**clone** P	Proceso replicado
	$\mid$	R	Proceso restricción
Ident. de objeto:			
I, J, K	$\rightarrow$	a, l	Nombres
	$\mid$	v	Valor
	$\mid$	x	Variable
Colección de métodos:			
M	$\rightarrow$	$[\, l_1 : (\, \widetilde{x_1}\,^a)\, P_1$ **&** $\ldots$ **&** $l_m : (\, \widetilde{x_m})\, P_m \,]$	
Mensajes:			
m	$\rightarrow$	$l : [\, \widetilde{I} \,]$	

[a] $\widetilde{y} \stackrel{\text{def}}{=} y_1, y_2, \cdots, y_n$

Figura 1: Gramática de PiCO.

A pesar de la amplia investigación y desarrollo en la programación por restricciones, una característica que comparte este paradigma con todos los lenguajes textuales es que su sintaxis resulta, todavía, alejada del lenguaje natural y más aún de la forma gráfica en que las ideas se crean con el pensamiento. Tomando otro punto de vista, la cantidad de código requerida (aunque no exagerada) para la modelación de un problema de alguna trascendencia resulta ser un impedimento para su comprensión. Es así como el grupo AVISPA diseñó el cálculo visual [AQRT98] para una versión pre-β del lenguaje de programación visual CORDIAL[9] [Tea98] (cálculo PiCO^{-1} expresado en forma gráfica, dada la complejidad que enmarca la especificación de todas las propiedades del cálculo PiCO de manera visual).

Como el código producido por el compilador CORDIAL-PiCO^{-1} no puede ser ejecutado directamente en una máquina real, se decide efectuar la implementación de un compilador referenciado como PiCO^{-1}-MAPiCO^{-1} [Cat99] que traduce el subconjunto elegido de código PiCO a código de una máquina virtual llamada MAPiCO (y referenciada aquí como MAPiCO^{-1}) [ABH98], para permitir la ejecución de un programa escrito en CORDIAL, luego de dos compilaciones: CORDIAL-PiCO^{-1}, PiCO^{-1}-MAPiCO^{-1}.

Posteriormente se efectuó reingeniería al lenguaje CORDIAL [JP02] de manera que se importó la implementación desde el paquete AWT hacia el paquete SWING, conjugando esta actividad con una documentación y estandarización del código fuente como una facilidad para abordar futuras propuestas. La representación visual del constructor condicional y la estructura a nivel de clases fueron los cambios desde el punto de vista sintáctico que se presentaron. Se buscó renovar esta interfaz, con el ánimo de modernizar la apariencia, en aras de que el lenguaje siga evolucionando y sirva como herramienta educativa.

El más reciente avance en cuanto a la máquina abstracta diseñada para ejecutar el código objeto CORDIAL es el que se mostró en [SL07], donde se consideraron las últimas modificaciones realizadas al cálculo PiCO y la extensión de instrucciones de la máquina a través de módulos dinámicos, que garantiza en un futuro no rediseñar la máquina abstracta. Asimismo se ofrece una alternativa de eliminación de variables locales de procesos que ya se han ejecutado por completo y que pueden no necesitarse más en el transcurso de la ejecución. Este trabajo discute la diferencia entre máquina virtual y máquina abstracta y exhibe varios ejemplos de cada tipo; así como también presenta formalmente el conjunto de instrucciones y la semántica mediante reglas de reducción.

El editor gráfico del lenguaje CORDIAL

Para editar un programa en el lenguaje de programación CORDIAL se requiere un entorno gráfico que permita la modelación de un algoritmo mediante un conjunto de objetos visuales. Este editor está contenido en lo que aparece en la Figura 2 como compilador de CORDIAL visual a CORDIAL textual.

[9] Primer lenguaje visual de restricciones de objetos concurrentes.

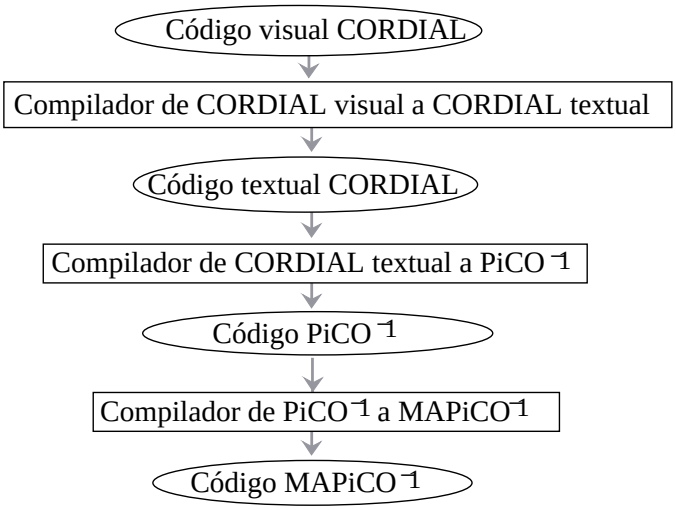


Figura 2: Etapas de la compilación en el lenguaje CORDIAL.

Este editor está compuesto por tres tipos de ventanas (presentadas en la Figura 3):

- Ventana de árbol de clases (cuadro superior izquierdo): listado de las clases que pueden ser utilizadas en el lenguaje, tanto las básicas o nativas, así como las definidas por el usuario.
- Ventana de edición de clases (cuadro superior derecho): ventana donde se definen las clases de usuario.
- Ventana de edición de métodos: Estas ventanas son empleadas para editar los métodos de usuario, que pueden ser: método principal (cuadro inferior izquierdo) y métodos de objetos base (cuadro inferior derecho).

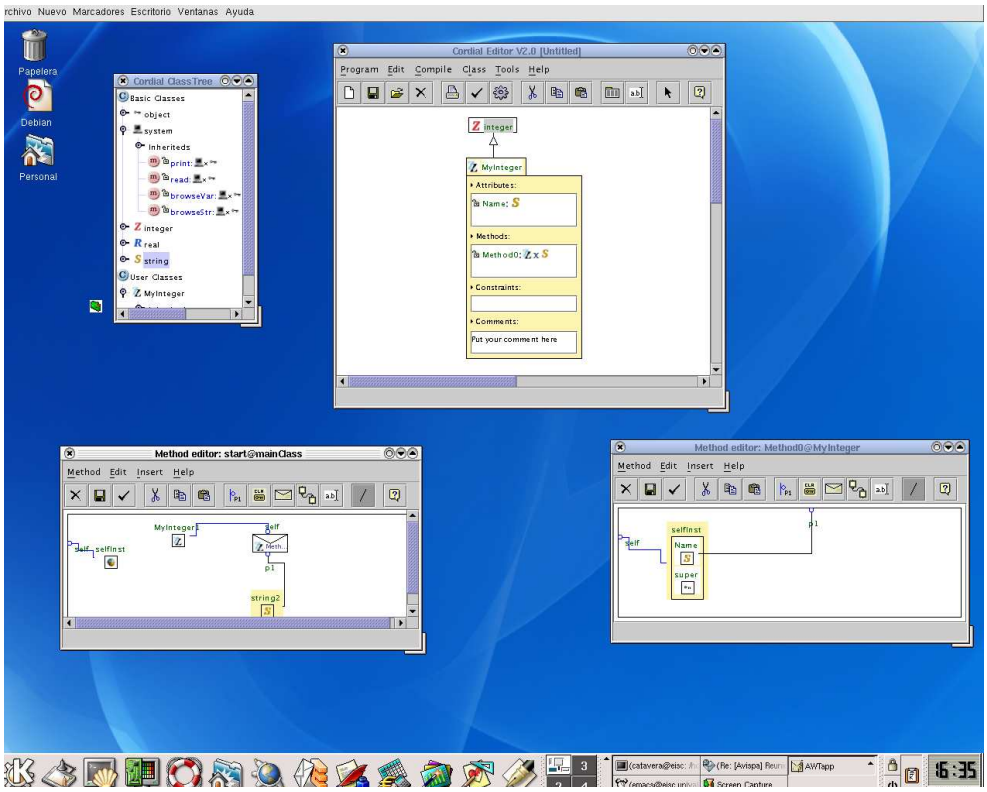


Figura 3: Vista de un programa CORDIAL.

Motivación en la creación de la interfaz visual de CORDIAL

Siguiendo las reglas fundamentales del diseño de interfaces gráficas[10], la implementación de un compilador del lenguaje visual CORDIAL a una parte del cálculo de imposición de restricciones orientado al objeto PiCO (PiCO^{-1}) fue muy importante para ofrecer al usuario final un excelente nivel de presentación, con características como: ayuda en línea, utilización de menús y demás propiedades ofrecidas por los entornos operativos provistos de ventanas.

Algunas de las características técnicas que motivaron la implementación de CORDIAL fueron: primero, la necesidad de acercar la sintaxis al programador, de manera que, se empleara un lenguaje visual mucho más didáctico. Segundo, ofrecer la posibilidad al programador de escoger el segmento y la profundidad en el código del programa a la hora de modelar el problema (mediante la forma de elección de objetos y clases). Y por último, el requerimiento de la implementación de una versión inicial de un compilador que contara con una porción de las características fundamentales del cálculo PiCO, para facilitar la ejecución y el análisis de pruebas.

[10] www.asktog.com/basics/firstPrinciples.html

Resultados de la etapa de análisis de eficiencia del compilador CORDIAL

Con base en las pruebas de eficiencia que el grupo AVISPA realizó al compilador CORDIAL-PiCO^{-1} (con problemas básicos como la función para calcular el factorial) 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 era aceptable. Desde otro ángulo, el tamaño del código no fue excesivo, como se demostró gracias a la creación de un compilador llamado CORTEX [Neb01], que transforma un código textual equivalente a CORDIAL al subconjunto elegido de cálculo PiCO (PiCO^{-1}).

Además, el trabajo de análisis de datos [GV01] descartó problemas en el diseño de los resolutores [GT98] o en la máquina virtual y mostró que el problema estaba, particularmente, en el código generado por el compilador CORDIAL-PiCO^{-1}, que en ciertos casos era código duplicado, inactivo o innecesario.

El rol de lo visual en el compilador CORDIAL-PiCO^{-1}

Se concluye que el problema fundamental es el esquema de traducción diseñado para generar código PiCO^{-1} que, aunque especifica una conversión directa de lenguaje visual al cálculo, no considera las propiedades formales del código objeto ni tiene una fase de optimización de código intermedio o la especificación de las reglas de traducción con optimización. Adicionalmente, las reglas de traducción no cuentan con una prueba de corrección.

Una característica que influencia de forma negativa la eficiencia del código compilado es la metodología de modelación textual con conjuntos[11] utilizada para el almacenamiento de la representación gráfica de un programa en el lenguaje visual CORDIAL y dibujada por el usuario desde la interfaz. Este código fuente no emerge de una especificación gramatical, lo que no permite un análisis dirigido por la sintaxis ni la generación de código mediante los mecanismos ampliamente utilizados de la teoría de la compilación de lenguajes textuales ni la de procesadores de códigos visuales. Con base en el trabajo de análisis, se puede identificar la necesidad de probar la corrección de las reglas de traducción para iniciar con una base sólida de estudio e implementación.

Solución al problema: el diseño y la elaboración del nuevo compilador GraPiCO

Propuesta de solución

Para solucionar el problema identificado en este documento se propone: La creación de GraPiCO: un nuevo cálculo visual, orientado al objeto y concurrente por restricciones. En este desarrollo se generarán unas reglas de traducción –con corrección– que se emplearán

[11] Cada objeto gráfico al estar compuesto –de manera lógica o física– por otros objetos gráficos (ejemplo: el objeto Casa contiene los objetos Ventana1, Ventana2 y Puerta), se representa por medio de un conjunto que contiene etiquetas: la primera, correspondiente al nombre del objeto gráfico más general, y el resto, a los nombres de los objetos gráficos relacionados con el primero mediante la relación de pertenencia. (ejemplo: {Casa, Ventana1, Ventana2, Puerta}).

para relizar traducción sin análisis léxico y sintáctico; la sintaxis se verificará en la etapa de edición de manera dinámica.

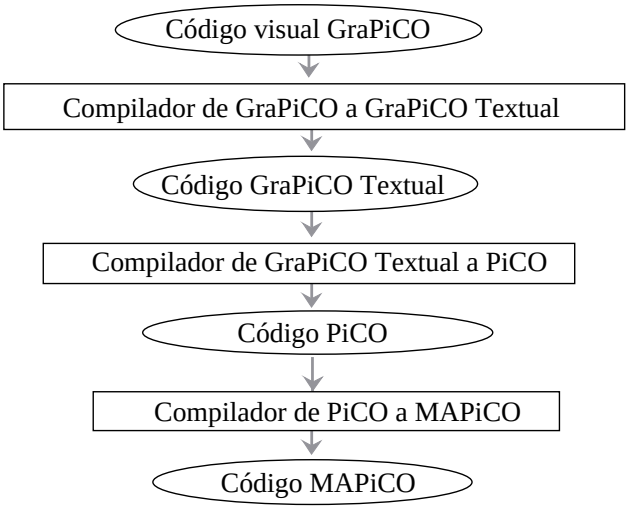


Figura 4: Etapas de la compilación en el nuevo lenguaje GraPiCO.

Objetivos

General

Diseño, implementación y demostración de corrección de GraPiCO: un nuevo cálculo visual, orientado al objeto y concurrente por restricciones.

Específicos

1. Fortalecer la herramienta de programación: compilador CORDIAL-PiCO^{-1}-MAPiCO^{-1}, mediante la creación de un nuevo cálculo visual GraPiCO que contendrá todo el potencial expresivo del cálculo PiCO.
2. Definir el nuevo cálculo visual GraPiCO, empleando una gramática para lenguajes visuales y con la presentación e implementación de las nuevas reglas de traducción mediante análisis dirigido por la sintaxis.
3. Probar la corrección de las reglas de traducción.
4. Diseñar un entorno gráfico de programación que no reitere las etapas de compilación.
5. Diseñar una forma de especificación sintáctica para el almacenamiento de los programas visuales y que permita su tratamiento formal.
6. Diseñar una manera de transformar los programas visuales GraPiCO en código de cálculo PiCO.

PARTE II

DESARROLLO

Capítulo 1

Nueva gramática del cálculo PiCO

La gramática mostrada en la Figura 1 corresponde a una representación diseñada para difundir la sintaxis del cálculo PiCO mas no para la creación del procesador de lenguaje respectivo. Por ello en el presente capítulo se presentará una nueva gramática equivalente que facilite la implementación de un compilador.

1.1. Gramática del cálculo PiCO en forma LL(1)

Con el objetivo de efectuar la implementación de una etapa de traducción dirigida por la sintaxis, se diseñó una gramática del código objeto (cálculo PiCO) que cumplió las condiciones de una definición lineal por la derecha con un símbolo de preanálisis LL(1).

A continuación, en la Figura 1.1 se presentará una gramática del cálculo PiCO en forma LL(1), organizada de manera que cada una de las partes izquierda de las producciones configure, en efecto, una etiqueta de un sintagma[1] de nuestro lenguaje; la organización también se efectuó con el fin de que cada una de las etiquetas empleadas como parte izquierda de varias producciones conformen un paradigma[2] de nuestro lenguaje. Todo lo anterior, siguiendo la definición de los dos ejes de Saussure presentados en [BS16].

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

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

- $S \rightarrow Lp$ **.**
- $Lp \rightarrow$ **(** $C\ PRp \mid C\ P$
- $Rp \rightarrow$ **|** $C\ PRp \mid$ **)**
- $C \rightarrow$ **clone** $\mid \epsilon$
- $P \rightarrow$

  (1) **local** $Lid\ Lp$
  - $Lid \rightarrow id\ Rid \mid \epsilon$
  - $Rid \rightarrow$ **,** $id\ Rid \mid$ **in**

  $id \rightarrow e \mid x$

  (2) **|** **(** ϕ_{sender} **,** $\delta_{forward}$**)** $\triangleright M$
  - $M \rightarrow$ **[** Lf

  $Lf \rightarrow f\ Rf \mid$ **]**

  $Rf \rightarrow$ **&** $f\ Rf \mid$ **]**

  $f \rightarrow e$ **:** **(** $Lx\ Lp$

  $Lx \rightarrow x\ Rx$

  $Rx \rightarrow$ **,** $x\ Rx \mid$ **)**

  (3) **|** $A\ T$
  - $A \rightarrow R\ \phi \mid I \triangleleft m$

  $R \rightarrow$ **ask** $\mid$ **tell**

  $I \rightarrow e \mid v \mid x$

  $m \rightarrow e$ **:** **[** LI

  $LI \rightarrow I\ RI$

  $RI \rightarrow$ **,** $I\ RI \mid$ **]**
  - $T \rightarrow$ **then** Lp

  (4) **|** ***O***

- Un programa S es una lista Lp.
- Lp es una lista de procesos. Un proceso tiene una condición C.
- Los procesos P separados por $\|$, representa concurrencia de procesos.
- La condición de replicación **clone** atribuye a los procesos persistencia.
- Existen cuatro tipos de procesos P:

  (1) Los id son variables x o nombres e, seguidos de procesos; **local** acota los id al ámbito del proceso, e **in** termina los id y encabeza el cuerpo del proceso.

  (2) Un objeto, cuya definición comienza con la definición de las restricciones de recepción (ϕ_{sender}) y delegación ($\delta_{forward}$); y termina por una colección de métodos (M).

  (3) Antecedente A Consecuente T
  - A puede ser: restricciones ϕ o un elemento I enviando el mensaje m, empleando el operador $\triangleleft$.

    R es imposición **tell** o consulta **ask**.

    Los elementos I son variables x, nombres e o valores v.
  - El consecuente es una lista de procesos.

  (4) ***O*** representa el proceso nulo.

Figura 1.1: Gramática LL(1) del cálculo PiCO.

Dado que las restricciones son parte fundamental del cálculo PiCO, requerimos su definición. Una restricción consiste en una lista de expresiones inorden de tres direcciones, separadas por caracteres conjunción ($\wedge$), donde cada expresión está compuesta por tres identificadores (que pueden ser: variables v, valores x o un literal **sender**), un operador de comparación ($\leq$, $\geq$, $<$, $>$, $=$, $\neq$) y un símbolo de operación ($+$, $-$, $\times$, $\div$, $\%$).

La siguiente definición gramatical G_ϕ presentada en la Figura 1.2 corresponde a la sintaxis de las restricciones en PiCO:

$Gramática\ G_\phi$:

- $\phi \rightarrow$ ($Restricción\ ListaRestricciones \mid Restricción$
- $ListaRestricciones \rightarrow \wedge\ Restricción\ ListaRestricciones \mid$)
- $Restricción \rightarrow$ **sender** $Comparación\ Posterior1$
  $\mid Identificador\ Comparación\ Posterior2$
- $Posterior1 \rightarrow\ Identificador\ Residuo1$
- $Posterior2 \rightarrow$ **sender** $Residuo1 \mid Identificador\ Residuo2$
- $Residuo1 \rightarrow\ Operador\ Identificador \mid \epsilon$
- $Residuo2 \rightarrow\ Operador\ Último \mid \epsilon$
- $Último \rightarrow\ Identificador \mid$ **sender**
- $Identificador \rightarrow\ v \mid x$ [a]
- $Comparación \rightarrow \leq \mid \geq \mid < \mid > \mid = \mid \neq$
- $Operador \rightarrow + \mid - \mid \times \mid \div \mid \%$

[a] variables o valores

Figura 1.2: Gramática de las restricciones en PiCO.

$Gramática\ G_{\phi_{sender}}$:

- $\phi_{sender} \rightarrow$ **sender** $Comparación\ Siguiente1$
  $\mid Identificador\ Comparación\ Siguiente2$
- $Siguiente1 \rightarrow\ identificador\ Resto1$
- $Siguiente2 \rightarrow$ **sender** $Resto1 \mid Identificador\ Resto2$
- $Resto1 \rightarrow\ Operación\ Identificador \mid \epsilon$
- $Resto2 \rightarrow\ Operación$ **sender** $\mid \epsilon$

Figura 1.3: Gramática de las restricciones de recepción.

De forma similar, también requerimos la especificación gramátical de dos tipos especiales de restricciones:

1. Restricciones de recepción (que, en adelante, llamaremos ϕ_{sender}) presentadas en la Figura 1.3.
2. Restricciones de delegación (las cuales desde ahora denominaremos $\delta_{forward}$)

Cada una de ellas consiste en una *restricción*, donde es obligatorio que un único campo de su expresión en tres direcciones genere el literal **sender** para ϕ_{sender}, y **forward** en el caso de $\delta_{forward}$.

La gramática de las restricciones de delegación $G_{\delta_{forward}}$ es el resultado de la sustitución del literal **sender** por el literal **forward**, en todas las producciones de $G_{\phi_{sender}}$.

$$G_{\delta_{forward}} = G_{\phi_{sender}}\{\mathbf{forward}\ /\ \mathbf{sender}\}$$

Con la nueva gramática del cálculo PiCO se crea la base para la implementación del correspondiente compilador y el punto de refencia para otros desarrollos.

Capítulo 2

El nuevo cálculo visual GraPiCO, sus bases formales

Dentro de los desarrollos para la implementación del nuevo compilador GraPiCO, se hace necesario diseñar un nuevo cálculo visual para el cálculo PiCO. A continuación se presenta primero el conjunto de nuevos constructores visuales de GraPiCO y su respectiva regla de traducción hacia un código textual; y segundo, se muestra la gramática completa del cálculo visual GraPiCO mediante una forma llamada Gsig (presentada en este capítulo).

Para entender mejor los dibujos y los mecanismos de especificación, se requiere la presentación de la definición del concepto de *expansión*.

Definición 1. *Llamaremos* **Expansión** *a la acción resultante de activar un ícono para visualizar su contenido. En la mayoría de entornos gráficos una expansión se logra presionando una cantidad determinada de veces un botón del ratón. En adelante se representará visualmente la expansión del ícono Ic, mediante el dibujo de la proyección de Ic hacia el conjunto de íconos que lo componen, como se presenta en la Figura 2.1.*

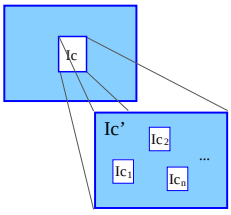


Figura 2.1: Ícono Ic' (representa la figura en su totalidad): expansión del ícono etiquetado con Ic en los íconos con etiquetas Ic_1, Ic_2 ... Ic_n dentro del ícono Ic'

2.1. Alfabeto del cálculo visual GraPiCO

El alfabeto de GraPiCO es presentado mediante las tablas de las Figuras 2.2 y 2.3.

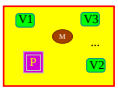
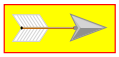
Nombre	Gráfico	Descripción
Programa		Un programa está compuesto por un grupo de procesos.
Clonación		La clonación es un costructor que acompañando los procesos les brinda la característica de la persistencia.
Proceso		Los procesos pueden ser: contextos, objetos o implicaciones.
Contexto		El contexto está compuesto de un conjunto de Identificadores (variables o métodos) acompañado de un programa.
Objeto		Un objeto es un constructor identificado por un nombre que está constituido por un grupo de métodos, y complementado por unas restricciones(de delegación y recepción) a definir en el objeto.
Método		Los métodos son elementos pertenecientes a un objeto que cumplen con la función de describir su comportamiento.
Implicación		Las implicaciones son constructores que cumplen la función de condicionales.
Ask		Este tipo de implicación representa una “consulta de restricciones” del antecedente y su consecuente es un programa o el proceso vacío.
Tell		Esta clase de implicación representa una “imposición de restricciones” del antecedente y su consecuente es un programa o el proceso vacío.
MsgSend		Esta implicación representa el “envío de un mensaje” en el antecedente desde un Identificador hacia un método, su consecuente puede albergar un programa o el proceso vacío.
Identificador		Los identificadores son literales que tienen un valor definido por la categoría utilizada.
Variable		La variable es un identificador que representa un valor cualquiera en casi cualquier Ambiente.
Argumento		El argumento es un identificador que representa un valor cualquiera, se diferencia de la variable en que su alcance está definido en el contexto de la ejecución de un método.

Figura 2.2: Constructores del cálculo visual GraPiCO (Parte 1).

Los dibujos que constituyen una clase de constructores aparecen en la tabla con el nombre en negrilla.

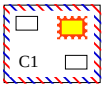
Nombre	Gráfico	Descripción
Recepción		Repositorio de un objeto donde se encuentra su conjunto de restricciones denominadas de recepción.
Delegación		Almacén de las restricciones de delegación de un objeto.
Puerto		Punto de conexión de las aristas que unen algunos constructores como Operadores y variables.
Sender		El emisor(*sender*) es un literal que representa el elemento que envía un mensaje, está presente en las restricciones de recepción.
Forward		El delegador(*forward*) es un literal que representa el elemento hacia donde se envían los mensajes que no pudieron ser contestados, está presente en las restricciones de delegación.
Operadores		Los operadores son los que calculan sobre términos numéricos o lógicos, están presentes en todo tipo de restricciones.
Restricciones		Las restricciones son los constructores base del cálculo visual GraPiCO, porque su función es acotar el comportamiento de los identificadores, a partir de la utilización de operaciones matemáticas y lógicas en expresiones de dos o tres direcciones; donde cada dirección es un identificador y están separados por un operador de comparación y uno matemático, o por sólo un operador de comparación. En GraPiCO existen tres categorías: restricciones genéricas, restricciones de recepción y restricciones de delegación.

Figura 2.3: Constructores del cálculo visual GraPiCO (Parte 2).

2.2. Sintaxis del cálculo visual GraPiCO

En esta sección se presenta una definición de la sintaxis del cálculo visual GraPiCO.

- Todo programa GraPiCO consta de un constructor de programa (rectángulo enmarcado en línea doble) que contiene un conjunto de procesos concurrentes (representados por rectángulos enmarcados en línea simple):

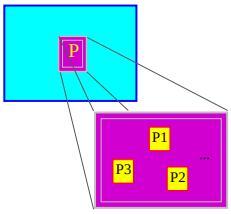


Figura 2.4: Programa P y su conjunto de procesos concurrentes $P1,\ P2,\ P3,\ \cdots$.

- Para determinar si un proceso tiene la propiedad de la replicación (clonación o persistencia) en la esquina inferior derecha de su ícono identificador se impone la respectiva señal (en este ejemplo un círculo).

Figura 2.5: Proceso $P2$ con la señal de replicación.

- Un proceso puede ser de tres tipos:
  - Constructor implicación: este constructor está compuesto de tres partes:
    - Antecedente: puede ser de dos tipos:
      - ◇ Consulta de restricciones: representado por un globo de diálogo que contiene un conjunto de restricciones –las cuales se evalúan para ejecutar el consecuente (explicado más adelante)–.
      - ◇ Imposición de restricciones: representado por un globo de diálogo estrellado; igual que la consulta, contiene un conjunto de restricciones –las que se intentan introducir al repositorio de restricciones– si esta operación tiene éxito entonces el consecuente es ejecutado.
      - Restricción: una restricción es modelada con un ícono carta que contiene una operación de comparación (compuesta de una expresión en dos o tres direcciones). Dado que dentro de las restricciones necesitamos referirnos al objeto que envía o al que le es enviado un mensaje, para tal motivo empleamos un recuadro sin etiqueta que corresponde al literal ***sender***, en el caso de las restricciones de recepción, o al literal ***forward*** para las restricciones de delegación.

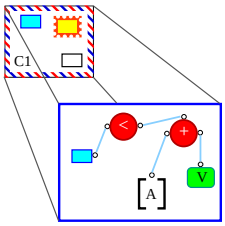


Figura 2.6: Restricción $C1$ y la operación de comparación $\boldsymbol{sender} < A + V$.

- Implicación: ícono (flecha) que enlaza el antecedente con el consecuente. Este ícono no contiene elemento alguno.
- Consecuente: programa (representado por un rectángulo delimitado en línea doble) que se ejecuta únicamente cuando el conjunto de restricciones que contiene el antecedente se ha evaluado y su evaluación arrojó una respuesta positiva.

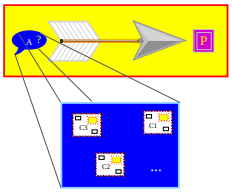


Figura 2.7: Constructor de consulta de restricciones con antecedente A (con restricciones $C1,\ C2,\ C3,\ \cdots$) y consecuente el programa P.

- Objetos: este constructor está compuesto por tres partes:
  - Definición de objeto: esta parte es una lista de métodos de objeto.
    - Cada método de un objeto (representado por un óvalo) está compuesto por una lista de argumentos (rectángulos enmarcados entre corchetes) y un programa.

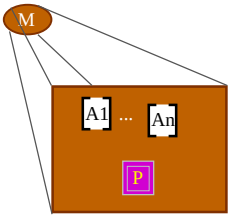


Figura 2.8: Método M con argumentos $A1,\ \cdots\ , An$ y el programa P.

- Restricciones de recepción: es el conjunto de restricciones que se deben cumplir para que el objeto al cual pertenecen atienda los mensajes.

- Restricciones de delegación: representa la guarda que procesa hacia donde irán los mensajes que deben ser delegados, cuando el objeto no cuenta con los métodos apropiados para atenderlos.

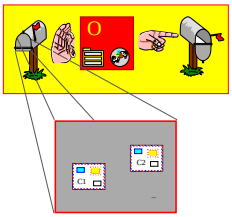


Figura 2.9: Constructor objeto con restricciones de recepción (representadas por el ícono del buzón cerrado), el objeto O y las restricciones de delegación (representadas por el buzón abierto).

- Nuevo ámbito para variables y nombres: este constructor está compuesto por tres partes:
  - Variables: conjunto de nuevas variables locales. Cada variable está representada por un rectángulo con una etiqueta.
  - Nombres: conjunto de nombres de métodos que intervendrán localmente. Estos están representados por un óvalo etiquetado.
  - Programa: lugar donde las variables y los nombres serán locales. Este constructor es repersentado por un rectángulo enmarcado con línea doble.

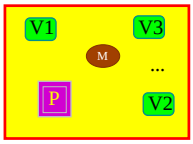


Figura 2.10: Constructor de nuevo ámbito con conjunto de variables $V1,\ V2,\ V3, \cdots$; conjunto de nombres $M, \cdots$; y programa P.

2.3. Congruencia estructural y relación de equivalencia del cálculo visual GraPiCO

Se define la congruencia estructural del cálculo GraPiCO de la misma manera que se hizo para el π-cálculo en [Mil93].

Definición 2. *(Congruencia estructural). En adelante se empleará el símbolo de relación de equivalencia ($\equiv$) para referirnos a la relación de congruencia más pequeña entre procesos GraPiCO satisfaciendo los siguientes axiomas:*

- Congruencia por α-conversión: Dos procesos son idénticos si únicamente difieren por un cambio de identificadores, una α-conversión.

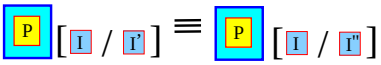


Figura 2.11: Procesos equivalentes luego de cambiar las apariciones de I por I'.

- Congruencia por concurrencia con proceso vacío: Dado que para todo proceso GraPiCO P que interactúe de forma concurrente con un proceso vacío es equivalente al proceso P y el operador de concurrencia es simétrico, tenemos que el conjunto de procesos con el constructor de concurrencia y el proceso vacío configura un monoide simétrico.

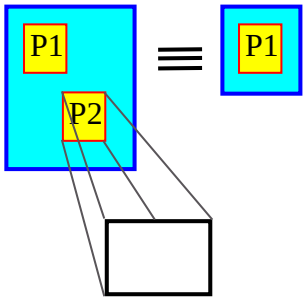


Figura 2.12: Proceso $P1$ actuando concurrentemente con un proceso vacío $P2$ y congruente con el proceso $P1$.

- Congruencia entre objetos: Dos objetos son congruentes si los conjuntos de restricciones y el conjunto de métodos son los mismos.

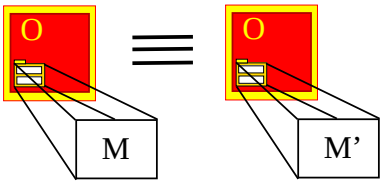


Figura 2.13: Objetos equivalentes con métodos y restricciones equivalentes.

- Congruencia del operador de replicación: La replicación de un proceso P se logra por medio de la operación concurrente entre el mismo proceso P y la ejecución de su replicación.

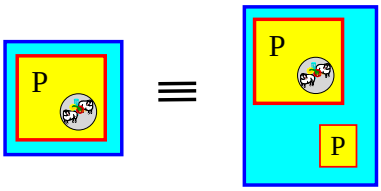


Figura 2.14: Proceso replicado P actuando concurrentemente con P y equivalente a P.

- Congruencia del operador nuevo ámbito:

  a) Congruencia del operador nuevo ámbito con programa vacío: La introducción de un nuevo identificador en un programa vacío es equivalente a un proceso vacío.

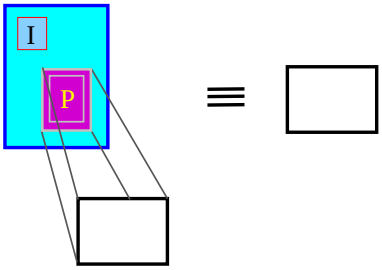


Figura 2.15: Declaración de nuevos identificadores en un proceso vacío y su equivalencia con el mismo.

  b) Congruencia del operador nuevo ámbito según el orden de declaración de los identificadores: El orden de la inclusión de nuevos identificadores no interfiere en la semántica de dos procesos equivalentes.

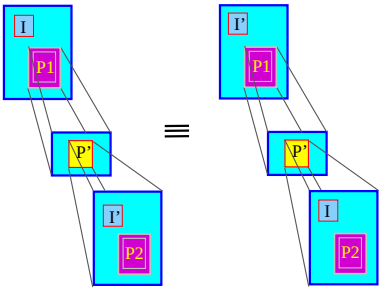


Figura 2.16: Procesos equivalentes con diferente orden de declaración de identificadores.

Capítulo 3

Especificación textual y sintaxis visual de GraPiCO

Este capítulo tratará el siguiente orden de ideas: 1) Introducción general, 2) Repaso de las gramáticas posicionales extendidas, 3) Propuesta de las Gsig y 4) Utilización de las Gsig en el nuevo cálculo visual GraPiCO.

3.1. Pertinencia de una especificación textual de GraPiCO

Para presentar la pertinencia de una especificación textual (es decir, qué tan oportuno y adecuado es detallar por medio de caracteres) cada uno de los constructores visuales de GraPiCO, se empleará la siguiente argumentación lógica:

i. Correspondencia entre constructores GraPiCO y sintagmas PiCO:

1. Se diseñó un constructor visual para cada no terminal de la gramática de PiCO.

2. Cada no terminal de esta gramática de PiCO corresponde a un sintagma.

---

3. Cada uno de los constructores visuales de GraPiCO corresponde a una representación visual de un sintagma del cálculo PiCO.

ii. Correspondencia entre especificación textual de GraPiCO y sintagmas PiCO:

3. Cada uno de los constructores visuales de GraPiCO corresponde a una representación visual de un sintagma del cálculo PiCO.

4. Se tiene una especificación textual de cada constructor visual GraPiCO.

---

5. Se tiene una especificación textual de la representación visual para cada uno de los sintagmas del cálculo PiCO.

Gracias a la conclusión *5.* se puede observar lo importante que es la especificación textual de cada uno de los constructores visuales GraPiCO, porque así se obtiene un código para los programas:

- Más fácil para almacenar.
- Formalmente tratable.
- Que almacena, además de toda la información de cada sintagma PiCO, toda la información visual requerida de los constructores visuales GraPiCO.
- Permite verificar la sintaxis.

A esta representación textual de los constructores GraPiCO se le llamará GraPiCO_Textual.

3.2. Breve panorama de la especificación textual de los VPL

La investigación sobre la formalización de lenguajes visuales de programación(VPL) se ha concentrado en dos líneas de investigación principales:

- Gramáticas para VPL, uno de sus grandes aportadores es el profesor Costagliola[1], cuyos trabajos más recientes se encaminan a la creación de *parsers* para VPL [CDFG01] cuando éstos utilizan un tipo de especificación gramatical denominada gramáticas posicionales extendidas [Cos00] (desde ahora GPE). La principal dificultad de esta propuesta radica en que los lenguajes diseñados siguiendo estas gramáticas deben ser simples, debido a que el análisis sintáctico propuesto requiere grandes cálculos y, como no está enfocado al almacenamiento, tampoco es fácil extraer un programa visual desde su representación textual; de igual manera, las demostraciones sobre propiedades formales del lenguaje exigen convertir el código en otro tratable matemáticamente.

- Semántica de VPL, un grupo de investigación sobresaliente en esta área es el del profesor Erwig[2] donde su desarrollo más novedoso acerca de los VPL es un estudio sobre la inferencia visual de persistencias [Erw06], empleando un trabajo anterior sobre la especificación semántica de VPL [Erw98b] en el cual presenta la importancia de una especificación textual de características como la semántica, para poder efectuar demostraciones en algún momento.

[1] http://www.dmi.unisa.it/people/costagliola/www/www/home/indice.htm
[2] http://web.engr.oregonstate.edu/∼erwig/

Desde lo aplicativo, recientes investigaciones se han interesado en el paradigma de programación Spreadsheet (estilo hoja de cálculo) como el presentado en [BAD+01], donde la especificación gramatical se facilita por la manera en la que se presenta la información y porque está orientado a la programación funcional donde la sintaxis no es muy extensa.

De otro lado, otro punto importante dentro de este paradigma son los adelantos encaminados a fortalecer la manipulación de la información contenida en las celdas mediante una extensión con programación lógica, no para manifestar su gramática sino más encaminado a la semántica; un trabajo que enfrenta la especificación textual de las relaciones en una hoja de cálculo se encuentra en [Phi07] y otra investigación sobre la manera de expresar textualmente los datos en las matrices encontradas en las hojas de cálculo y su posterior tratamiento para evaluar procesos como la unificación se muestra en [PN08]. Al ver el panorama, no hay grandes cambios en las formas de especificación gramatical desde las GPE, y la utilización de otra implica la creación de nuevas maneras de *parsing* o la disminución de la expresividad del lenguaje. Igualmente, los programas visuales, por ser dibujos, requieren una representación textual no sólo para su almacenamiento y posterior extracción, sino para realizar los procesos de los cuales sólo se conocen métodos textuales. Esta necesidad no está contemplada de forma clara por las especificaciones gramaticales existentes.

En sus inicios, el grupo AVISPA[3] (capítulos Universidad del Valle y Universidad Javeriana de Cali-Colombia) empleó las GPE para la implementación de GraPiCO[4]; pero en el desarrollo encontró que estos mecanismos no se ajustaban a los requerimientos, dado que representaban un análisis sintáctico complicado y el código de almacenamiento era difícil de tratar. Por todas las razones expuestas, se diseñó una forma de especificación para VPL denominada Gsig, que permitió el empleo de dibujos, la utilización de mecanismos de *parsing* conocidos y probados como el descendente predictivo para gramáticas LL(1) y la obtención de un código textual para almacenar los programas. De esta manera los aportes que se presentarán de las Gsig son: un mecanismo de especificación gramatical que utiliza métodos de análisis sintáctico conocidos, flexibilidad en los íconos empleados y la obtención de un código textual para la representación de los programas visuales que permitan desde su dibujo el almacenamiento y viceversa, y su tratamiento formal, como por ejemplo, comprobación de la semántica y diseño de reglas de traducción.

[3] Grupo de investigación en **A**mbientes **VIS**uales de **P**rogramación **A**plicativa.

[4] Lenguaje visual con base en el cálculo PiCO (π Calculus and Concurrent Objects, desarrollado por el grupo AVISPA).

3.3. Especificación sintáctica de lenguajes visuales

En esta sección se recuerda la definición de GPE, una especificación sintáctica para VPL.

3.3.1. Gramáticas posicionales extendidas

Las GPE presentadas en [Cos00] extienden las gramáticas libres de contexto para los lenguajes textuales hacia gramáticas para los visuales, mediante nuevas relaciones adicionales a la concatenación. Para establecer estas relaciones se requiere un conjunto de identificadores de relación y un evaluador pictórico para verificar su comportamiento.

Las GPE están compuestas por producciones de la forma:

$$A \rightarrow \overbrace{X_1\ R_1\ X_2\ R_2\ \cdots X_{m-1}\ R_{m-1}\ X_m}^{Lista\ de\ símbolos\ relacionados}\ \Delta, \Gamma \tag{3.1}$$

Donde:

- A: es un símbolo no_terminal.
- $Lista\ de\ símbolos\ relacionados$: es una sucesión de símbolos del alfabeto y de relaciones que un par de símbolos consecutivos cumplen, por ejemplo, $X_1\ R_1 X_2$ representa que X_1 está relacionado con X_2 mediante la relación R_1.
- Δ: es un conjunto de *reglas para sintetizar* los valores de los atributos sintácticos de A a partir de los atributos de $X_1,\ X_2, \cdots,\ X_m$.
- Finalmente, Γ: es un conjunto de *triplas de inserción* de la forma $(N, Cond, \Delta)$. Estas últimas se emplean para insertar de forma dinámica nuevos símbolos terminales en la frase de entrada durante la etapa de análisis sintáctico.
  Donde:
  - N: es el símbolo terminal a ser insertado.
  - $Cond$: es una precondición que se debe cumplir para insertar N.
  - Y, Δ: es la regla usada para calcular los atributos de N a partir de los atributos de $X_1,\ X_2,\ \cdots,\ X_m$.

Por ejemplo, en la Figura 3.1 se presenta un objeto visual compuesto por un conjunto de íconos con etiquetas $Ic_1, Ic_2, \ldots, Ic_n$ dentro del ícono Ic'. Se desea definir una gramática que lo describa, además de un mecanismo para almacenar los atributos de los íconos. Una GPE que represente el conjunto de íconos de la Figura 3.1 se muestra en la Figura 3.2.

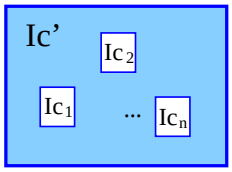


Figura 3.1: Conjunto de íconos con etiquetas Ic_1, Ic_2, ... , Ic_n dentro del ícono Ic'.

1. $Ic' \rightarrow GrupoIc'$	1. Un grupo de íconos es un conjunto de íconos Ic'.
2. $Ic \rightarrow Ic_1 \qquad \Delta(Ic = Ic_1)^a$	
3. $Ic \rightarrow \vdots$	2.- 4. Todo ícono Ic_i (del tipo i) es un ícono genérico Ic.
4. $Ic \rightarrow Ic_n \qquad \Delta(Ic = Ic_n)$	
5. $GrupoIc' \rightarrow GrupoIc\ \langle \mathbf{Junto}^b \rangle\ Ic$ $\Delta(GrupoIc' = f(GrupoIc,\ Ic))^c$ $\Gamma : \{(CONTEXTO^d, \lvert GrupoIc \rvert > 0^e,$ $CONTEXTO = g(GrupoIc,\ Ic)^f)\}$	5. Un grupo de íconos al lado de un ícono es un grupo de íconos.
6. $GrupoIc' \rightarrow GrupoIc\ \langle \mathbf{R}^g \rangle\ CONTEXTO$ $\Delta(GrupoIc' = CONTEXTO)$	6. Un grupo de íconos y un CONTEXTO es un grupo de íconos.
7. $GrupoIc' \rightarrow Ic$ $\Delta(GrupoIc' = Ic)$	7. Un ícono genérico es un grupo de íconos.

[a] Los atributos de Ic_1 son tomados por Ic.
[b] **Junto** en una relación cumplida por cualquier par de íconos que comparten la misma ventana.
[c] Los atributos de $GrupoIc'$ son calculados mediante la función f.
[d] Terminal abstracto insertado dinámicamente en la frase de entrada durante el análisis sintáctico.
[e] Condición que asegura que un grupo de íconos está compuesto por, al menos, un ícono.
[f] Los atributos de CONTEXTO son calculados mediante la función g.
[g] **R** es una relación que siempre es satisfecha por cualquier par de símbolos.

Figura 3.2: Gramática posicional extendida de un grupo íconos: $Ic_1, Ic_2, \ldots, Ic_n$ dentro del ícono Ic'.

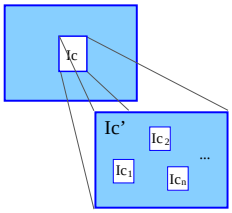


Figura 3.3: Ícono Ic' (representa la figura en su totalidad): expansión del ícono etiquetado con Ic en los íconos con etiquetas Ic_1, Ic_2, ... , Ic_n dentro del ícono Ic'.

Si se quiere modelar el caso de la Figura 3.3 en GPE, una nueva relación *Expansión* es requerida. Una gramática posicional para este caso es presentada en la Figura 3.4.

- $Ic^{\wr} \rightarrow Ic \ \langle \textbf{expansión} \rangle \ Ic'$

Figura 3.4: Gramática posicional extendida del ícono $Ic^{\wr}$, compuesto de Ic y su expansión Ic'

El grupo AVISPA usó este tipo de especificación sintáctica en el editor de CORDIAL, aplicación que fue plasmada en [CP99] obedeciendo a la dinámica de una época en la que el desarrollo de aplicaciones había conocido un giro interesante. Del enfoque de décadas pasadas de ofrecer herramientas que, si bien podían resolver el problema, resultaban difíciles y pesadas de usar, se estaba pasando a un periodo en el que, sin descuidar la potencia de la herramienta, se provee un ambiente que facilita su manejo por personas no expertas en informática. Se abrió así toda una línea de sistemas ergonómicos, que basan su comunicación con el usuario en una interfaz gráfica. Como en todo lenguaje de programación, en CORDIAL se requirió un analizador sintáctico que identifique cuándo un programa está o no bien construido según una especificación gramatical. El analizador sintáctico de CORDIAL se compone de dos subanalizadores:

- Analizador de lectura de entrada: el cual recibe un archivo de texto plano con las coordenadas de los elementos que conforman un programa y lo convierte en una estructura de datos hashtable en Java, con los elementos organizados de acuerdo con las relaciones existentes entre ellos. El archivo de entrada se construye captando las características de cada objeto o componente en su correspondiente subeditor, por medio de métodos implementados con este objetivo como getClases() –el que retorna el conjunto de clases definidas en el subeditor de clases–, y getListRef() –el que entrega la lista de métodos y restricciones en los determinados subeditores–.
  Este analizador se implementó utilizando el generador de analizadores Java Bison.

- Analizador de la salida: este proceso toma la salida del primer analizador y se encarga de establecer si las relaciones entre los objetos son válidas o no a partir de la gramática posicional de CORDIAL. Las relaciones empleadas para modelar un programa CORDIAL son: $incluye$, $horizontal$, $vertical$, $toca$, $incluyetoca$, $debajode$, $encimade$ y $linea$.

  Según estas relaciones se tomaron como errores sintácticos:

  - El solapamiento de elementos.
  - Construcciones no permitidas dentro del lenguaje.
  - Unión no permitida por medio de líneas que pertecen a conjunciones diferentes dentro de un mismo condicional.
  - Existencia de elementos no permitidos como: líneas con origen pero sin destino, puertos sin asociarse a un elemento o instancias con puertos sin líneas.

  La relaciones entre objetos se encuentran almacenadas en la estructura hashtable resultante del primer analizador.

  Para construir este analizador se usó el generador de analizadores JavaCup.

Aunque el analizador sintáctico total hace el trabajo requerido, está compuesto de dos subanalizadores que hacen trabajo repetido; de otro lado, la cantidad de estructuras de datos requeridas para almacenamiento es muy grande comparada con la cantidad de constructores en los programas visuales, ver Figura 3.5. Por estos motivos resulta interesante una forma de especificar los lenguajes visuales que evite reiterar procesos y que necesite poco espacio.

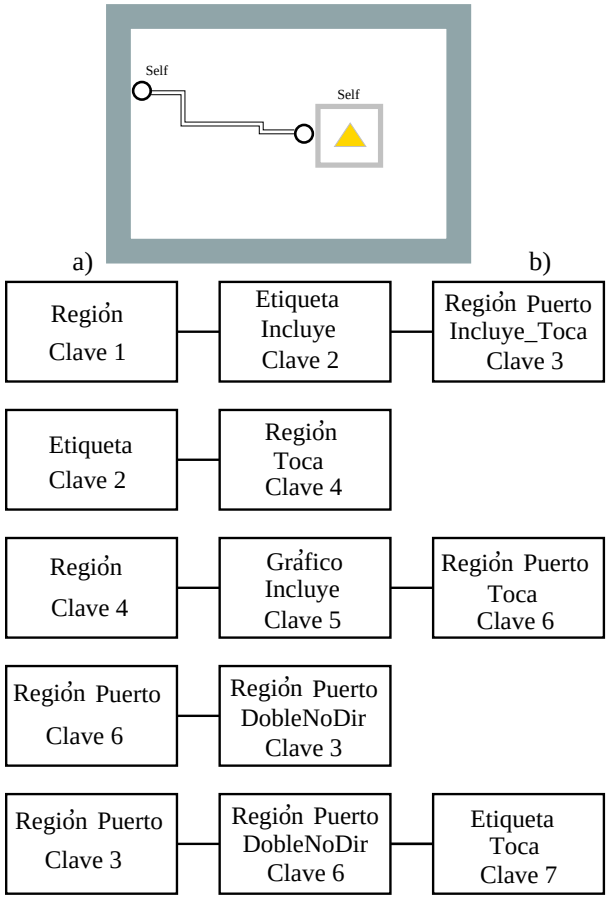


Figura 3.5: a) Programa CORDIAL y b) su estructura hashtable.

En seguida se presentará la nueva propuesta por la cual pueden modelarse los lenguajes visuales.

3.4. Alternativa de especificación sintáctica: gramática de sistemas de icónicos generalizados - Gsig

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 marcas [5][FNFS06]; entre los más famosos se encuentran el HTML [Rag05] (Hyper Text Markup Language) para la descripción de páginas web, el OWL [owl04] (Web Ontology Language) para presentación de ontologías y los esquemas XML para marcado descriptivo [XML06] (Extensible Markup language). Hasta el momento se han empleado los lenguajes de marcas para tratamiento (ver [Bol01]) y transformación de código textual, como el trabajo en [Hir06]; desde el punto de vista gráfico, también se han usado para representar y traducir hojas de cálculo, como el que se presentó en [EK05]. Sin embargo, 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 debido a la complejidad y la extensión del código textual requerido para modelar un programa visual.

En esta sección se hará una introducción a la teoría de la composición de los sistemas icónicos y su mecanismo de especificación. Luego, empleando marcas se presentarán formalmente las Gsig y ejemplos de su uso.

3.4.1. Composición de los sistemas icónicos

Empleando la teoría presentada en [Cha90], donde se introduce el término sistema icónico[6].

Todo ícono generalizado está compuesto de dos partes:

1. Parte física: La información de la imagen como tal. Ejemplos de este tipo de datos son: la localización del archivo almacenando la gráfica y la localización en el espacio de trabajo.
2. Parte lógica: El significado de la imagen. Aquí está comprendida información como: a qué clase de sintagma[7] corresponde el ícono y por cuáles íconos está compuesta la imagen.

Esta caracterización de la composición de los componentes visuales será utilizada en la definición formal de las Gsig presentada en la sección 3.4.2.

3.4.2. Casos de especificación de constructores visuales

Existen dos clases de constructores visuales:

1. Simple: Se efectúa a un ícono en particular sin tomar su entorno.
2. Compuesto: Realizada al conjunto de constructores visuales simples que lo conforman.

A continuación se presentan las Gsig mediante los mecanismos para lograr los dos tipos de especificaciones de constructores visuales.

[5]Lenguajes de marcas: 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.

[6]Conjunto estructurado de íconos generalizados relacionados.

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

Especificación de un constructor visual simple

La especificación de un constructor visual simple X en términos de Gsig se logrará efectuando los siguientes pasos:

1. Abstrayendo su información física como:
   - Una coordenada de referencia del ícono del constructor, expresado en términos de una pareja ordenada $(x - y)$.
   - El camino de la gráfica del ícono del constructor en el medio de almacenamiento. A este camino se seguirá denominando $Path(X)$.
   - La etiqueta dada por el usuario al ícono en el programa. A esta etiqueta se llamará $Name(X)$.
2. Abstrayendo su información lógica o, lo que es lo mismo, la especificación de la expansión del constructor que se está especificando. En adelante, para referirse a la expansión del ícono de un constructor GraPiCO etiquetado con X, se empleará la notación $Ex(X)$. De igual forma, se utilizará $\mathcal{E}[\![X]\!]$ como la función semántica que entrega la especificación textual del constructor con etiqueta X ó la especificación del constructor mismo.
3. Y organizando la información obtenida en 1. y 2. entre un par de símbolos de delimitación ($Sd_1 Sd_2$)[8] como se muestra en la ecuación 3.2:

$$\mathcal{E}[\![X]\!] = S^{9}\ Sd_1\ \overbrace{(x-y)\ Path(X)}^{Parte\ Física}\ \sim\ \overbrace{Name(X)}^{Etiqueta}\ :\ \overbrace{\mathcal{E}[\![Ex(X)]\!]}^{Parte\ Lógica}\ Sd_2 \tag{3.2}$$

Especificación de un constructor visual compuesto

La especificación de un constructor visual compuesto $X_1, \cdots, X_n$ dentro de un sintagma se efectúa al listar las diferentes especificaciones de cada constructor, separadas por un símbolo de sincronización $\widehat{Sc_i}$ que identifica la relación i entre un par de constructores; lo anterior, encerrado entre un par de símbolos de delimitación Sd_1 y Sd_2, como se presenta en la ecuación 3.3:

$$\mathcal{E}[\![X_1, \cdots, X_n]\!] = Sd_1\ \mathcal{E}[\![X_1]\!]\ \widehat{Sc_1} \cdots \widehat{Sc_{n-1}}\ \mathcal{E}[\![X_n]\!]\ Sd_2 \tag{3.3}$$

3.4.3. Definición del proceso de construcción de las Gsig a partir de la especificación de los constructores visuales

Luego de la definición del proceso de especificación de los constructores visuales, se muestra la caracterización de las Gsig mediante la definición 3.

[8] Ejemplos de Sd_1 y Sd_2 son { }, ( ), [ ], ⟨ ⟩.

[9] En ocasiones se requiere el empleo de un símbolo de sincronización S o símbolos de delimitación para establecer condiciones en los constructores e identificarlos de manera única.

Definición 3. *Una Gsig es una gramática independiente de contexto, donde las producciones pueden ser de tres formas:*

1. $X \rightarrow \mathcal{E}[\![Y]\!]$	1. *Generación de una especificación.*
2. $X \rightarrow \mathcal{E}[\![Ex(Y)]\!]$	2. *Generación de una Parte lógica.*
3. $X \rightarrow (\ Coordenadas\ )\ Imagen$	3. *Generación de una Parte física.*

Al emplear la definición 3 se construye la producción más general para un constructor visual, presentada en la ecuación 3.4.

$$X \rightarrow Y \ \sim \ Etiqueta \ : \ Z \tag{3.4}$$

En la ecuación 3.4 el símbolo $\rightarrow$, en efecto, contiene dos connotaciones en su función semántica:

1. El constructor visual X está compuesto de: Y, no_terminal que representa la parte física, el símbolo de sincronización '$\sim$'; un símbolo de identificación denominado $Etiqueta$, el símbolo de sincronización '**:**' y finalmente, por Z, no_terminal que corresponde a su parte lógica.

2. La $Expansión$ del ícono representado por X tiene como resultado lo que produce Z.

De otro lado, gracias a la estructura de la producción mostrada en la ecuación 3.4 son modelados diversos casos de constructores visuales en donde alguno de sus elementos es nulo, estos casos se presentan en la Figura 3.6.

1. Constructor sin parte física, $Y \rightarrow \epsilon$.

   $X \rightarrow \sim\ Etiqueta\ :\ Z$

2. Constructor sin etiqueta.

   $X \rightarrow Y\ \sim : Z$

3. Constructor sin parte lógica, $Z \rightarrow \epsilon$.

   $X \rightarrow Y\ \sim\ Etiqueta\ :$

1. Elemento gráfico del que se conoce su ubicación por referencia a otro elemento o porque siempre está en el mismo lugar.
2. Objeto visual que no tiene asociada una etiqueta, ya sea porque el lenguaje le asigna una o porque no la requiere.
3. Ícono que internamente no está compuesto de otros elementos visuales; en otras palabras, la función *Expansión* no retorna elemento alguno. Este caso se emplea para modelar los literales visuales.

Figura 3.6: Casos de constructores visuales modelados por las Gsig.

3.4.4. Ejemplo de utilización de las Gsig

En esta sección se mostrará un ejemplo de especificación con las Gsig, retomando los ejemplos de los constructores visuales de la Figuras 3.1 y 2.1.

Gramática de un constructor visual compuesto

- Al considerar el caracter '**,**' como el símbolo de sincronización separador de los íconos que comparten una ventana.
- Empleando la ecuación 3.3, la definición 3 y la ecuación 3.4.

  Se obtiene la gramática para el constructor visual compuesto expuesta en la Figura 3.1, presentada en la ecuación 3.5.

$$Ic' \rightarrow \sim Etiqueta\_De\_Ic'\ :\ \mathcal{E}[\![Ic_1]\!]\ ,\ \cdots\ ,\ \mathcal{E}[\![Ic_n]\!] \tag{3.5}$$

- Finalmente, al resolver las especificaciones de los íconos en la ecuación 3.5 y empleando los símbolos de delimitación '**(**' y '**)**' se tendría la gramática de la ecuación 3.6.

$$Ic' \rightarrow \sim Etiqueta\_De\_Ic'\ :\ (\ ParteFísica\_De\_Ic_1\ \sim\ Etiqueta\_De\_Ic_1\ :\ ParteLógica\_De\_Ic_1\ ,\ \cdots\ ) \tag{3.6}$$

Gramática de un constructor visual simple

- A través de la ecuación 3.2, la definición 3 y la ecuación 3.4.

  Se consigue la gramática para el constructor visual mostrado en la Figura 2.1, presentada en la ecuación 3.7.

$$Ic \rightarrow \ ParteFísica\_De\_Ic \sim Etiqueta\_De\_Ic \ : \ \mathcal{E}[\![\ Ex(Ic)\ ]\!] \tag{3.7}$$

- Y al emplear la producción que se indica en la ecuación 3.5.

  Se llegaría a la gramática de la ecuación 3.8.

$$Ic \rightarrow \ ParteFísica\_De\_Ic \sim Etiqueta\_De\_Ic \ : \ Ic' \tag{3.8}$$

Gramática de la composición de un constructor visual simple y un constructor visual compuesto

Luego de la construcción de la gramática en la sección 3.4.4 del constructor visual compuesto presentado en la Figura 3.1; y la gramática en la sección 3.4.4, del constructor visual simple que se enmarca en la Figura 2.1, se muestra en la ecuación 3.9 la gramática del par de constructores visuales, a partir de las gramáticas planteadas en las ecuaciones 3.6 y 3.8.

$$\begin{aligned} &Ic \rightarrow ParteFísica\_De\_Ic \sim Etiqueta\_De\_Ic \ : \ Ic' \\ &Ic' \rightarrow \sim Etiqueta\_De\_Ic' \ : \\ &\quad (\ ParteFísica\_De\_Ic_1 \ \sim \ Etiqueta\_De\_Ic_1 \ : \ ParteLógica\_De\_Ic_1\ ,\ \cdots\ ) \end{aligned} \tag{3.9}$$

Posteriormente, una transformación que favorece la comprensión de la gramática presentada en la ecuación 3.9, es la de la inclusión de un no_terminal por cada ícono del constructor visual compuesto $\{Ic_1, \cdots, Ic_n\}$, además de la adición de un identificador (que correspondería al símbolo de sincronización S mencionado en la ecuación 3.2) para cada categoría de íconos, así: 1 para Ic_1, 2 para Ic_2, ... y n para Ic_n.

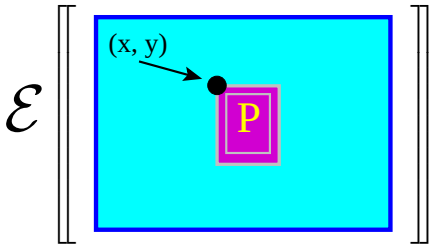
La gramática final es expuesta en la ecuación 3.10.

$$\begin{aligned} &Ic \rightarrow ParteFísica\_De\_Ic \sim Etiqueta\_De\_Ic \ : \ Ic' \\ &Ic' \rightarrow \sim Etiqueta\_De\_Ic' \ : \ (Ic_1\ ,\ \cdots\ , Ic_n\ ) \\ &Ic_1 \rightarrow \mathbf{1}\ ParteFísica\_De\_Ic_1 \ \sim \ Etiqueta\_De\_Ic_1 \ : \ ParteLógica\_De\_Ic_1 \\ &\quad \vdots \\ &Ic_n \rightarrow \boldsymbol{n}\ ParteFísica\_De\_Ic_n \ \sim \ Etiqueta\_De\_Ic_n \ : \ ParteLógica\_De\_Ic_n \end{aligned} \tag{3.10}$$

En la siguiente sección se emplearán las Gsig para la especificación de cada uno de los constructores visuales del cálculo visual GraPiCO.

3.5. Lista de fórmulas de especificación de GraPiCO hacia GraPiCO_Textual

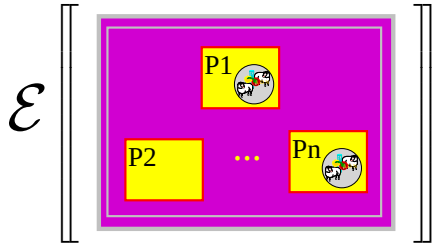
1. Especificación de un programa: Está compuesta por las coordenadas del punto superior izquierdo y el camino de la dirección física del archivo que contiene la imagen del ícono que representa el programa, luego por el caracter virgulilla (∼) seguido del nombre del programa y el caracter dos puntos (:); todo encerrado entre caracteres de llaves ({,}).



$$\mathcal{E}[\![\ ]\!] = \{(x-y)\ Path(P) \sim Name(P) : \mathcal{E}[\![Ex(P)]\!]\}$$

Figura 3.7: Especificación de un programa.

2. Especificación de procesos concurrentes: Consiste en una lista de las especificaciones de cada uno de los procesos, separadas por el caracter barra vertical (|).



$$\mathcal{E}[\![\ ]\!] = (\ \mathcal{E}[\![P_1]\!]\ |\ \mathcal{E}[\![P_2]\!]\ |\cdots|\ \mathcal{E}[\![P_n]\!]\ )$$

Figura 3.8: Especificación de procesos concurrentes

3. Especificación de una condición de persistencia: También llamada de replicación, se reduce a verificar si está presente o no el ícono de clonación; si éste está presente, la especificación es el caracter asterisco (*), de lo contrario es la cadena vacía.

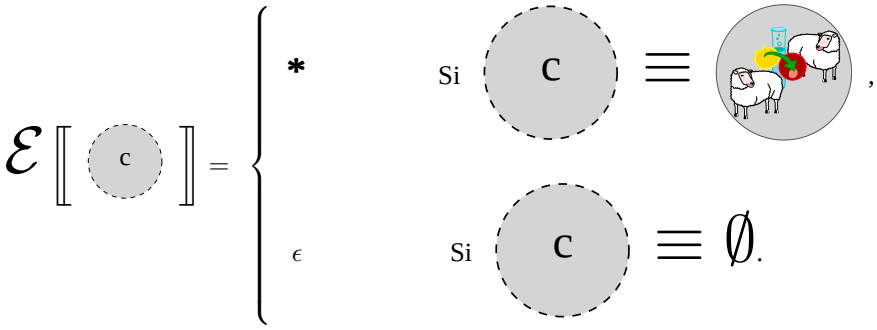


Figura 3.9: Especificación de una condición de persistencia

4. Especificación de un proceso: Está conformada por la especificación de la condición de persistencia, seguida de las coordenadas del punto superior izquierdo y el camino de la imagen del ícono que representa el proceso, posteriormente por el caracter virgulilla (∼) y el nombre del proceso seguido del caracter dos puntos (:); en último lugar, se encuentra la especificación de la expansión del ícono del proceso.

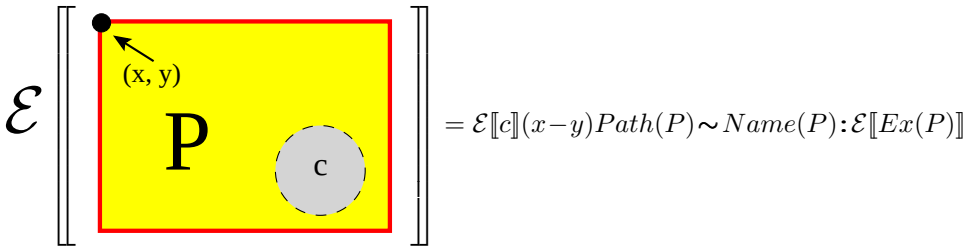


Figura 3.10: Especificación de un proceso

Definición 4. *Un* **grupo de variables** *en la creación de ámbito, consiste en el conjunto de todas las variables presentes en este último. De manera similar, un* **grupo de nombres** *en la creación de ámbito, consta del conjunto de todos los nombres presentes en el mismo. Como se observa en la Figura 3.11.*

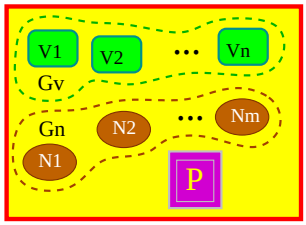
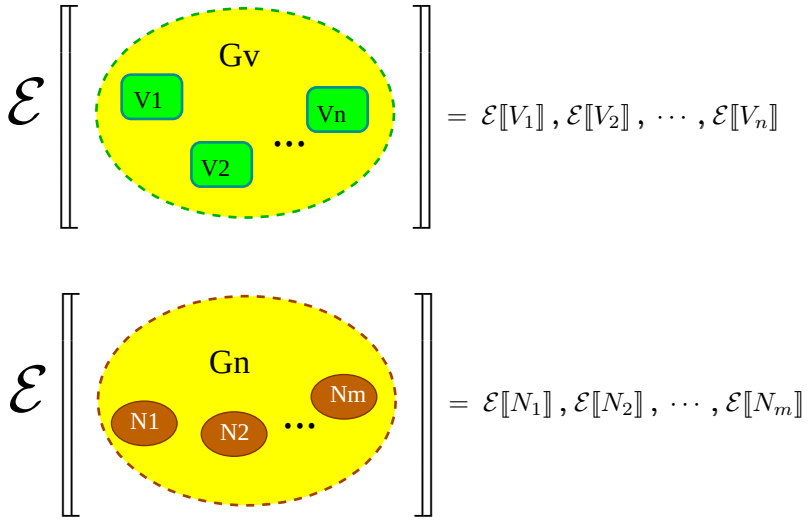


Figura 3.11: Grupo de variables $G_v = \{V_1,\ V_2,\ \cdots,\ V_n\}$ y grupo de nombres $G_n = \{N_1,\ N_2,\ \cdots,\ N_m\}$ en la creación de ámbito para grupos de variables y nombres.

5. Especificación de un grupo de variables o grupo de nombres en la creación de un nuevo ámbito: Está compuesta por la lista de las Especificaciones de cada uno de sus elementos, separadas por caracteres coma (**,**).

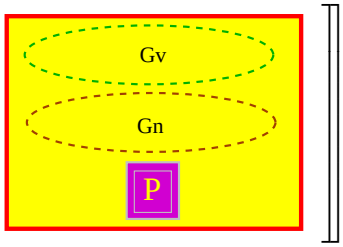


$$\mathcal{E}[\![\;]\!] = \mathcal{E}[\![V_1]\!] \, , \mathcal{E}[\![V_2]\!] \, , \cdots , \mathcal{E}[\![V_n]\!]$$

$$\mathcal{E}[\![\;]\!] = \mathcal{E}[\![N_1]\!] \, , \mathcal{E}[\![N_2]\!] \, , \cdots , \mathcal{E}[\![N_m]\!]$$

Figura 3.12: Especificación de un grupo de variables o de un grupo de nombres en la creación de un nuevo ámbito.

6. Especificación de una creación de ámbito para variables y nombres: Se trata de la especificación del grupo de variables, seguida por la especificación del grupo de nombres ,y finalmente, por la especificación del programa; donde los tres componentes están separados por caracteres punto y coma (**;**) y delimitados por caracteres corchetes (**[ ]**).



$$\mathcal{E}[\![\;]\!] = [\, \mathcal{E}[\![G_v]\!] \, ; \mathcal{E}[\![G_n]\!] \, ; \mathcal{E}[\![P]\!] \,]$$

Figura 3.13: Especificación de una creación de ámbito para variables y nombres.

7. Especificación de un método: Está compuesta por las coordenadas del punto superior izquierdo y el camino y la etiqueta de la imagen del ícono que representa el método; terminado esto, por el caracter virgulilla (**∼**) seguido de la expansión y la correspondiente especificación del ícono del constructor.

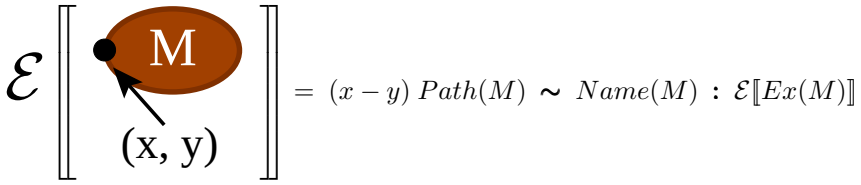


Figura 3.14: Especificación de un método.

8. Especificación de una variable: Consta de las coordenadas del punto superior izquierdo y el camino de la imagen del ícono que representa la variable; culminado lo anterior, por el caracter virgulilla (∼) seguido del nombre de la variable (etiqueta dada por el usuario).

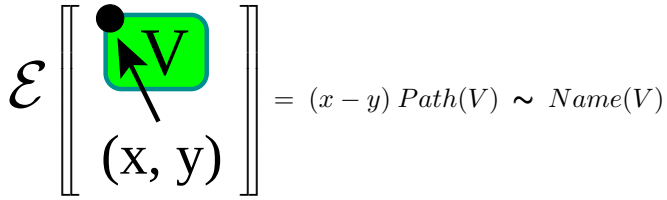


Figura 3.15: Especificación de una variable.

9. Especificación de un objeto: Está compuesta por la especificación de la expansión del ícono que representa al objeto, luego por el caracter triángulo hacia la derecha (⊳), y finalizada, por la especificación de la expansión del campo definición de objeto.

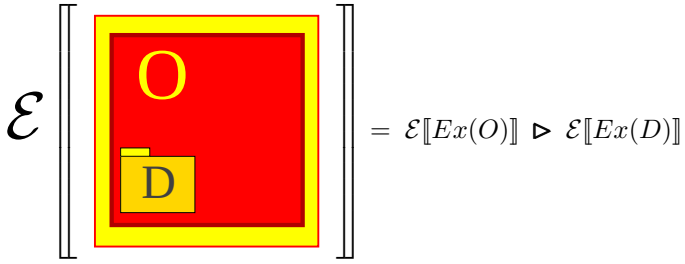


Figura 3.16: Especificación de un objeto.

10. Especificación del campo definición de objeto: Se trata de la especificación del subcampo métodos, todo lo anterior, encerrado entre caracteres de corchetes (**[ ]**).

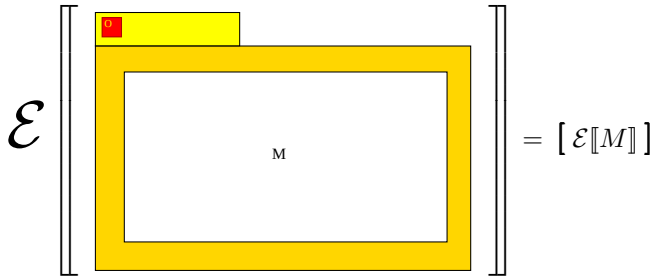


Figura 3.17: Especificación de definición de objeto.

Definición 5. *Una* **lista de elementos** *de objeto, consiste en la lista de métodos del campo métodos de un objeto. Gráficamente, se puede ver en la Figura 3.18.*

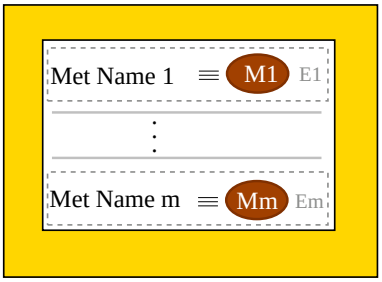


Figura 3.18: Lista de elementos (métodos) de objeto: $L_M = \{E_1, \cdots, E_m\}$.

11. Especificación de lista de elementos de objeto: Está compuesta por la lista de las especificaciones de cada uno de los elementos, separadas por el caracter et (*ampersand*) (**&**).

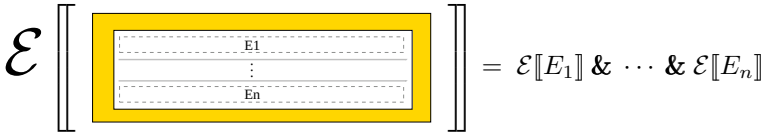


Figura 3.19: Especificación de una lista de elementos de objeto.

12. Especificación de un método en un objeto: Consiste en el nombre del método seguido del caracter dos puntos (**:**), y finalmente, por la especificación del ícono que representa el método.

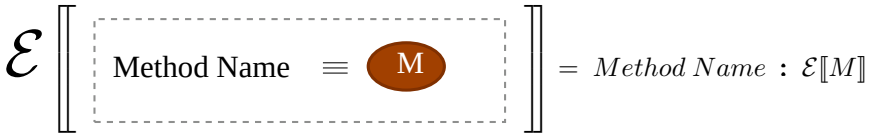


Figura 3.20: Especificación de un método.

Definición 6. *Un* **grupo de argumentos** *en un método, consiste en el conjunto de todos los argumentos presentes en la expansión del ícono del método. Se puede observar en la Figura 3.21.*

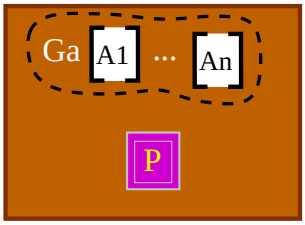


Figura 3.21: Grupos de argumentos $G_a = \{A_1, \cdots, A_n\}$.

13. Especificación de grupo de argumentos en un método: Comprende la lista de las especificaciones de cada uno de los argumentos separados por caracteres coma (**,**), encerrada entre paréntesis (**( )**).

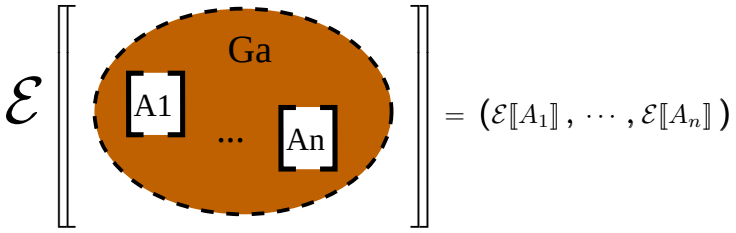


Figura 3.22: Especificación de un grupo de argumentos en un método.

14. Especificación del cuerpo de un método: Consiste en la especificación del grupo de argumentos presentes en la expansión del ícono que representa el método, seguida por la especificación del programa que conforma el cuerpo del método.

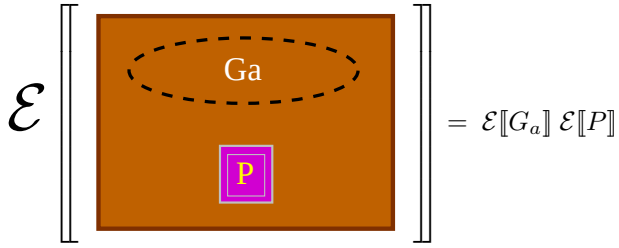


Figura 3.23: Especificación del cuerpo de un método.

15. Especificación de un argumento: Consta de las coordenadas del punto superior izquierdo y el camino de la imagen del ícono que representa el argumento; posteriormente, por el caracter virgulilla ($\sim$) seguido del nombre del argumento.

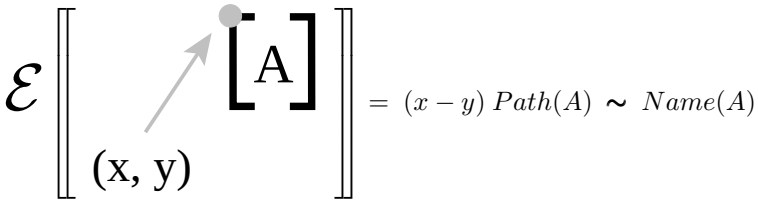


Figura 3.24: Especificación de un argumento.

16. Especificación de las restricciones de un objeto: Está compuesta por la especificación de la expansión del ícono que representa las restricciones de recepción, luego, por la especificación de la expansión del ícono que representa las restricciones de delegación, los dos elementos anteriores separados por el caracter coma (**,**), y para terminar, todo encerrado entre caracteres de paréntesis (**( )**).

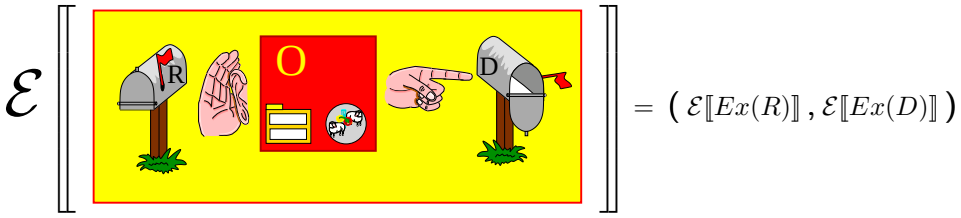


Figura 3.25: Especificación de restricciones de objeto.

17. Especificación de un antecedente en una implicación: Se resuelve evaluando tres casos:

    a) Si el antecedente es una consulta de restricciones, la especificación es el símbolo de sincronización '?', seguido de la especificación de la expansión del ícono del antecedente.

    b) En caso de que el antecedente sea una imposición de restricciones, la especificación será el símbolo de sincronización '!', continuado por la especificación de la expansión del icono correspondiente al antecedente.

    c) Si por el contrario, el envío de un mensaje es el antecedente, el resultado de la especificación será la especificación de la expansión del ícono correspondiente al envío del mensaje.

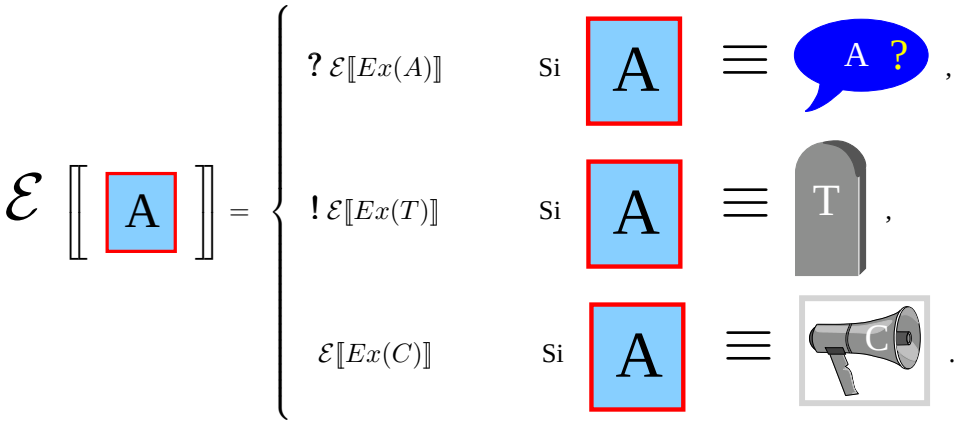


Figura 3.26: Especificación de un antecedente en una implicación.

18. Especificación de una implicación: Se trata inicialmente de la especificación del ícono que corresponde al antecedente (que puede ser una imposición o consulta de una restricción o el envío de un mensaje) y después, por la especificación del ícono del programa que corresponde al consecuente; las anteriores dos especificaciones separadas por un caracter coma (,) y todo entre caracteres de delimitadores angulares $(\langle\,\rangle)$.

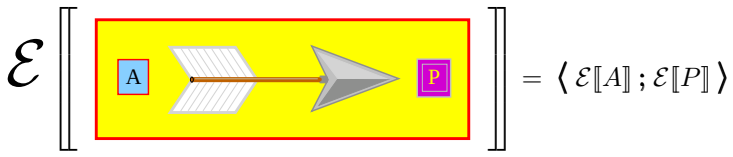


Figura 3.27: Especificación de una implicación.

Definición 7. *Un* **grupo de Relaciones en una aplicación** *del envío de un mensaje, consiste en el conjunto de las relaciones creadas entre los identificadores (que pueden ser nombres, variables o Valores) y los Parámetros de un método, para efectuar la aplicación de dicho procedimiento a los argumentos.*

Ejemplo: Como se presenta en la Figura 3.28,

El método M tiene:

- los argumentos $A_1,\ A_2,\ \cdots, A_n$. Y,
- como Parámetros, los Identificadores $I_1,\ I_2,\ \cdots, I_n$.

Esto es la aplicación $M[A_1,\ A_2,\ \cdots,\ A_n](I_1,\ I_2,\ \cdots,\ I_n)$.

Para lo cual se efectuarán las siguientes sustituciones: $[M/A_1]I_1,\ \cdots, [M/A_n]I_n$.

De la lista de sustituciones se observa la siguiente relación $[M/A_i]I_j \ \Rightarrow\ A_i\ R\ I_j$.

De lo anterior resulta $G_r \overset{\text{def}}{=} \{(A_i, I_j) : A_i\ R\ I_j\}$.

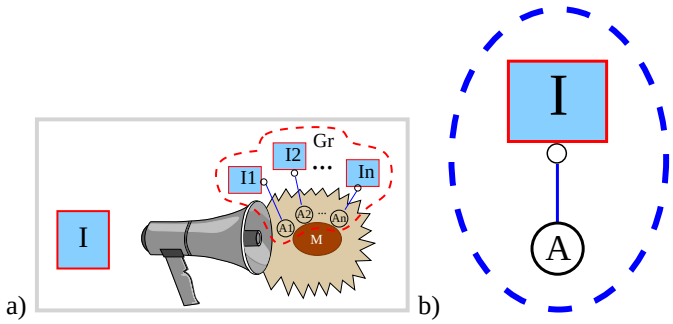


Figura 3.28: a) Grupo de Relaciones en una aplicación: $G_r = \{R_1 = (A_1, I_1),\ R_2 = (A_2, I_2),\ \cdots,\ R_n = (A_n, I_n)\}$, b) Relación de aplicación.

19. Especificación de un grupo de Relaciones en una aplicación:

Dado que:

a) La expansión del ícono que representa un método M muestra un grupo de argumentos G_a.

b) La especificación de G_a da como resultado una sucesión **(** $[\![A_1]\!],\ \cdots,\ [\![A_n]\!]$ **)**.

Entonces la especificación de un grupo de Relaciones $G_r = \{R_1 = (A_1, I_1),\ R_2 = (A_2, I_2),\ \cdots,\ R_n = (A_n, I_n)\}$ es igual a la lista de las especificaciones de cada una de las Relaciones de aplicación separadas por el caracter coma (**,**) y encerradas entre caracteres de corchetes **([ ])**.

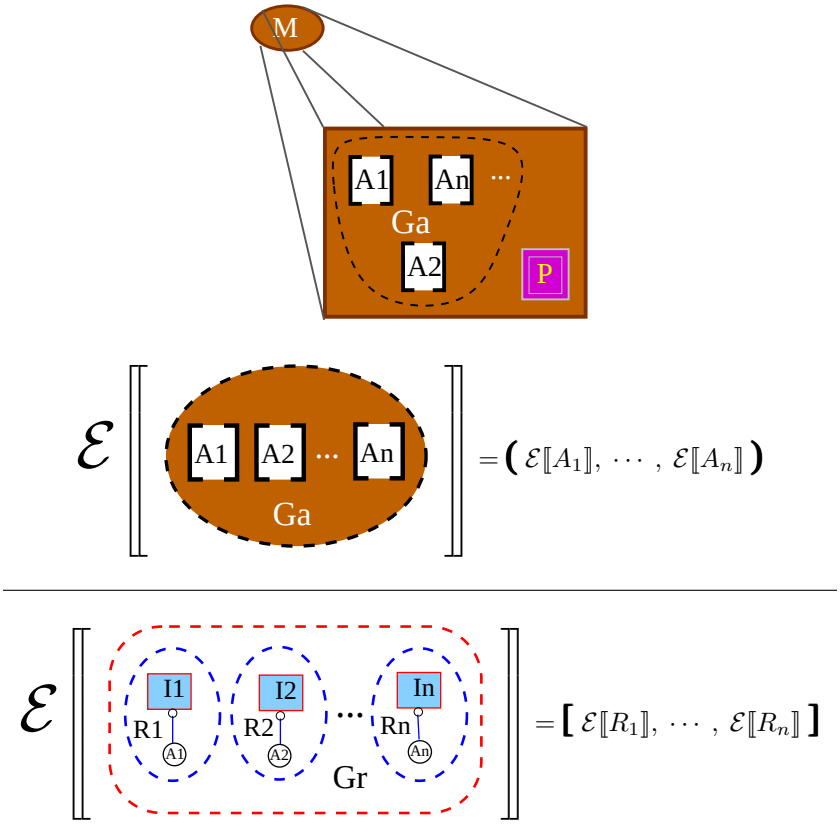


Figura 3.29: Especificación de un grupo de Relaciones en una aplicación.

20. Especificación de una relación de aplicación: Está conformada por la especificación del puerto encerrado entre parentesis $(\,())$ y por la especificación del Identificador respectivo.

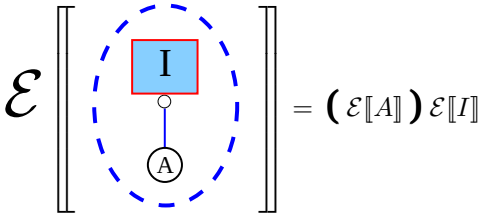


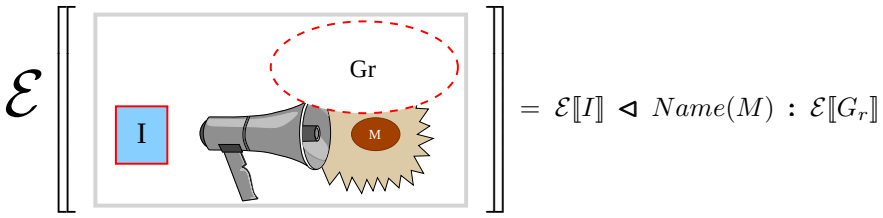
Figura 3.30: Especificación de una relación de aplicación.

21. Especificación de un puerto de un método: Está compuesta por el punto central y el camino del ícono que representa el constructor, posteriormente, por el caracter virgulilla ($\sim$) y el nombre asociado con el puerto (nombre del argumento en el método).

$$\mathcal{E}[\![\ \textcircled{A}\ ]\!] = (x-y)\ Path(A)\ \sim\ Name(A)$$

Figura 3.31: Especificación de un puerto de método.

22. Especificación de un envío de mensaje: Consiste en la especificación del ícono de la estructura del cálculo que Envía el mensaje, seguida del caracter triángulo hacia la izquierda ($\lhd$) y el nombre del método empleado para la comunicación, luego por el caracter dos puntos (**:**) y de último, por la especificación del grupo de Identificadores de Parámetros.



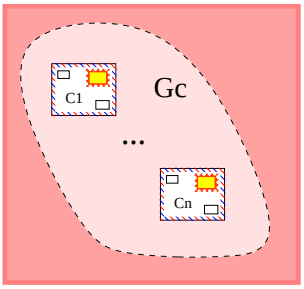
$$= \mathcal{E}[\![I]\!] \lhd Name(M)\ :\ \mathcal{E}[\![G_r]\!]$$

Figura 3.32: Especificación de un envío de mensaje.

Definición 8. *Un* **grupo de restricciones** *Está formado por el conjunto de todos los íconos de restricción (carta) presentes en:*

- *Un objeto en alguno de los campos:*
  - *Restricciones de recepción.*
  - *Restricciones de delegación.*
- *Alguna de las expansiones de los siguientes antecedentes de implicación:*
  - *Imposición de restricciones.*
  - *Consulta de restricciones.*

Por ejemplo, la Figura 3.33:



grupo de restricciones $G_c = \{C_1, \cdots, C_n\}$:

Figura 3.33: Ejemplo de grupo de restricciones.

23. Especificación de un grupo de restricciones: Está conformado por la lista de las especificaciones de cada restricción separadas por el caracter de conjunción (∧).

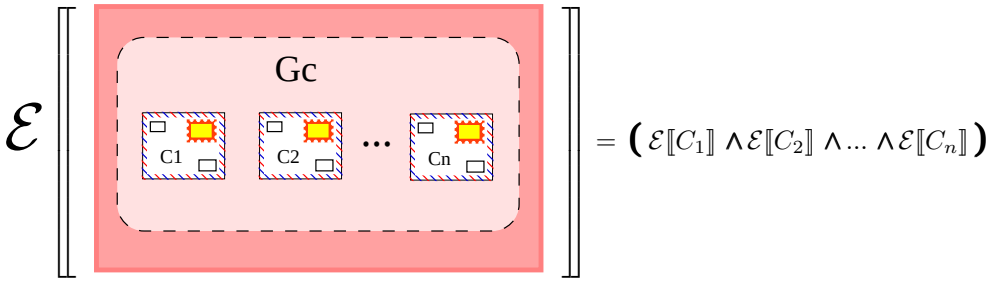
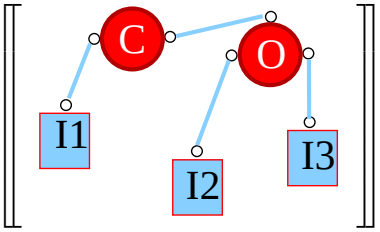
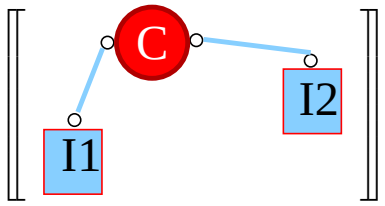


Figura 3.34: Especificación de un grupo de restricciones

24. Especificación de la expansión de una restricción (carta): Tiene como resultado la organización de manera inorden de la especificación de cada uno de sus componentes.

a) $\mathcal{E}$ 

$= \mathcal{E}[\![I_1]\!]\ \mathcal{E}[\![C]\!]\ \mathcal{E}[\![I_2]\!]\ \mathcal{E}[\![O]\!]\ \mathcal{E}[\![I_3]\!]$

b) $\mathcal{E}$ 

$= \mathcal{E}[\![I_1]\!]\ \mathcal{E}[\![C]\!]\ \mathcal{E}[\![I_2]\!]$

Figura 3.35: Especificación de la expansión de una restricción: a) restricción compuesta por una comparación C y una operación O, b) restricción compuesta por una comparación C.

25. Especificación de los componentes comparación y operación en una restricción.

a) $\mathcal{E}$ 

$= (x - y)\ Path(C)\ \sim\ Name(C)$

$Name(C) =$ *símbolo empleado por el usuario para la comparación* $>, = \cdots$.

b) $\mathcal{E}$ 

$= (x - y)\ Path(O)\ \sim\ Name(O)$

$Name(O) =$ *Símbolo empleado por el usuario para la operación* $+, * \cdots$.

Figura 3.36: Especificación de los componentes de una restricción: a) Comparación C, b) Operación O.

26. Especificación del literal visual receptor o delegador en una restricción de objeto.

$$\mathcal{E}\ [\![\ \boxed{\text{L}}\ ]\!] = (x - y)\ Path(L)\ \sim\ Name(L)$$

$$Name(L) = \begin{cases} \textbf{sender} & \text{si } L \text{ está presente en una restricción de recepción,} \\ \textbf{forward} & \text{si } L \text{ está presente en una restricción de delegación.} \end{cases}$$

Figura 3.37: Especificación de un literal receptor o delegador en una restricción de objeto.

3.6. Gsig de GraPiCO

Se presenta la gramática Gsig del cálculo visual GraPiCO en forma LL(1), resultado de la generalización de las reglas de especificación textual de los constructores GraPiCO para cualquier programa GraPiCO.

Como se aprecia en la Figura 3.38 y en las producciones presentadas en la Figura 3.39, un programa está compuesto de una parte física: una imagen (referenciando el programa) con su respectiva localización, una etiqueta (identificación que el usuario le impone al ícono del programa), y una parte lógica: conformada por el cuerpo del programa, que es a su vez, un conjunto de procesos concurrentes. De esta forma, para representar gramaticalmente lo anterior, dado que la expansión del no_terminal $Programa$ debería mostrar su parte lógica, el símbolo $\rightarrow$ tiene dos significados: el usual en gramáticas BNF y el de la $Expansión$.

Para presentar la concurrencia (procesos en una misma ventana) se emplea el operador |. La utilización de | como símbolo de sincronización, obedece al empleo que se hace de este caracter en la gramática original de PiCO y el deseo de continuar con la notación. No existe ambigüedad entre el símbolo aquí usado, ni el usual en las gramáticas BNFE (para agrupación de varias partes derechas en una parte izquierda), porque al ser la gramática de GraPiCO LL(1), cuando se emplea | como símbolo de sincronización de especificaciones siempre aparecerá como parte de PRIMERO[10] de una producción; mientras que, cuando | se emplea como separador BNFE, nunca se encontrará al inicio de una parte derecha de alguna producción.

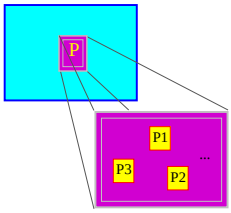


Figura 3.38: Programa GraPiCO compuesto por los procesos concurrentes: P_1, P_2, P_3...

[10]Conjunto de componentes léxicos que operan como los primeros símbolos de una o más cadenas generadas a partir de la producción.

$Programa \rightarrow \{\ ParteFísica\ \sim\ Etiqueta\ :\ ParteLógicaPrograma\ \}$

- $ParteFísica \rightarrow (\ Coordenadas)\ Imagen$
- $ParteLógicaPrograma \rightarrow$

  $(\ Clonación\ Proceso\ ListaProcesos \mid Clonación\ Proceso$

  - $ListaProcesos \rightarrow \mid\ Clonación\ Proceso\ ListaProcesos\ \mid\ )$
  - $Proceso \rightarrow ParteFísica\ \sim\ Etiqueta\ :\ ParteLógicaProceso$
  - $Clonación \rightarrow *\ \mid \epsilon$

Figura 3.39: Sección de gramática para programa.

Los procesos dentro de un programa GraPiCO pueden ser de tres clases:

1. Creación de un ámbito para un conjunto de variables y nombres de métodos de objetos.
2. Definición de un objeto.
3. Empleo de una implicación, donde si se puede ejecutar con éxito el proceso situado como antecedente, entonces, se procesará el programa localizado en el consecuente.

Ejemplos de los tres tipos de procesos se pueden ver en la gráfica 3.40 donde se emplea una “lupa” para apreciarlos más de cerca.

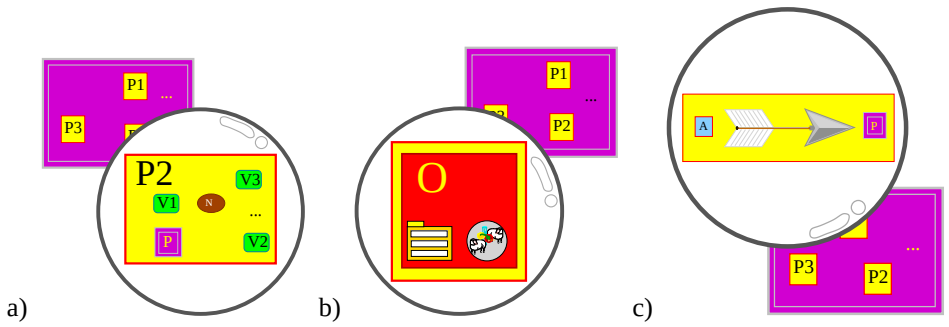


Figura 3.40: Ejemplos de los tipos de procesos vistos “de cerca”: a) Definición de ámbito, b) Objeto, c) Implicación.

De la misma forma que existe el operador **clone** en PiCO, en GraPiCO se puede emplear el operador icónico presentado en la Figura 3.41 a), posicionándolo sobre el ícono del proceso al que se quiera replicar, como se muestra en la Figura 3.41 b).

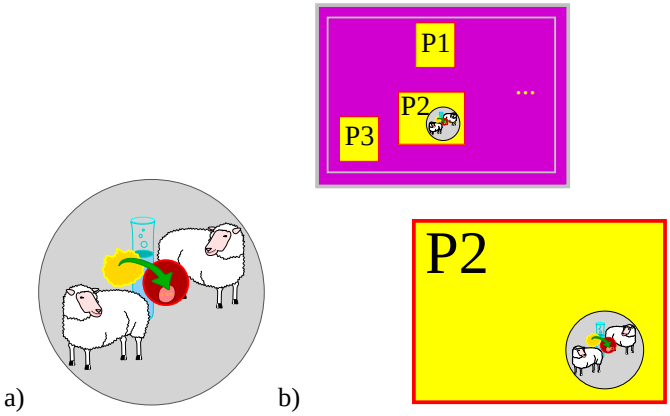


Figura 3.41: Replicación de procesos: a) Operador icónico para la replicación, b) Proceso P_2 con replicación.

La representación visual de la modelación de un ámbito (ver Figura 3.42), se efectua mediante la localización de un conjunto de íconos de variables o nombres de métodos, acompañando al ícono de un programa en una misma ventana; esto significa que estas variables y nombres de métodos son locales al programa referido.

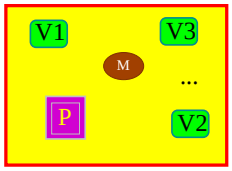


Figura 3.42: Proceso: definición de ámbito. Las variables: V_1, V_2, V_3... y el mensaje M, son locales al programa P.

$ParteLógicaProceso \rightarrow$
[$ListaNombres$; $ListaVariables$; $Programa$] | ... **sigue en la Figura 3.46**

- $ListaNombres \rightarrow Nombre\ RestoNombres \mid \epsilon$
  - $RestoNombres \rightarrow$, $Nombre\ RestoNombres \mid \epsilon$
  - $Nombre \rightarrow ParteFísica \sim NombreMétodo$
- $ListaVariables \rightarrow Variable\ RestoVariables \mid \epsilon$
  - $RestoVariables \rightarrow$, $Variable\ RestoVariables \mid \epsilon$
  - $Variable \rightarrow ParteFísica \sim NombreVariable$

Figura 3.43: Sección de gramática para el proceso definición de ámbito.

Otro tipo de proceso son los objetos. Están compuestos de dos partes:

1. Restricciones de objeto, donde se modela la forma como actuará el objeto mediante la imposición de restricciones. Ver la Figura 3.44 a).
2. Definición de objeto, donde se presenta la lista de métodos del mismo. Ver la Figura 3.44 b)
   Por su parte, los cuerpos de método están compuestos de una lista de argumentos y un programa, como se aprecia en la Figura 3.45.

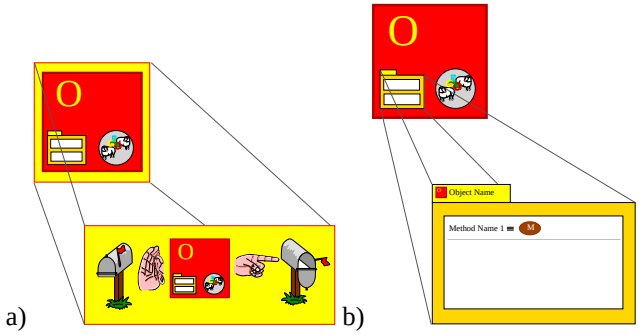


Figura 3.44: a) Restricciones en objeto, b) Definición de objeto.

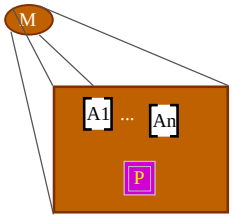


Figura 3.45: Método M compuesto de la lista de argumentos: $A_1, \ ... \ , \ A_n$ y el programa P

$ParteLógicaProceso \rightarrow$

viene de la Figura 3.43

$\vdots$

$|$ ($RestriccionesRecepción$, $RestriccionesDelegación$) $\triangleright$ [$ListaMétodos$]

$\vdots$

sigue en la Figura 3.52

Figura 3.46: Sección de gramática para el proceso objeto.

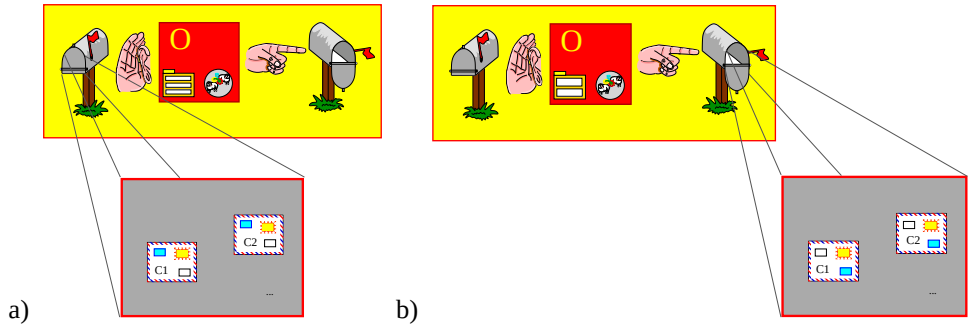
Las restricciones de objeto se dividen en dos tipos:

1. Para recepción, donde se establece la condición a cumplir por los objetos que envíen algún mensaje, para que sean atendidos. Un ejemplo se presenta en la Figura 3.47 a), donde el objeto que tenga estas restricciones para recepción, únicamente atenderá a los objetos que cumplan las restricciones C_1 y C_2.

2. Para delegación, donde se especifica que algún otro objeto que contenga estas restricciones en el campo de las restricciones de recepción, atenderá los mensajes que no pudieron ser contestados.

   La restricción de delegación representa una medida de seguridad para saber hacia dónde los procesos mensaje deben ser delegados cuando no exista un método apropiado para atenderlos. Involucra la creación de una nueva instancia emisora del mensaje original; pues antes de que el mensaje sea enviado, el nuevo objeto emisor es obligado a satisfacer la restricción de delegación, de manera que el objeto al que se le delegará el mensaje tenga la oportunidad de contestar.

   Para evitar el envío ciclico de mensajes el sistema sólo reenviará un mensaje cuando se pueda asegurar que el nuevo objeto sea diferente del inicial. Ver Figura 3.47 b).



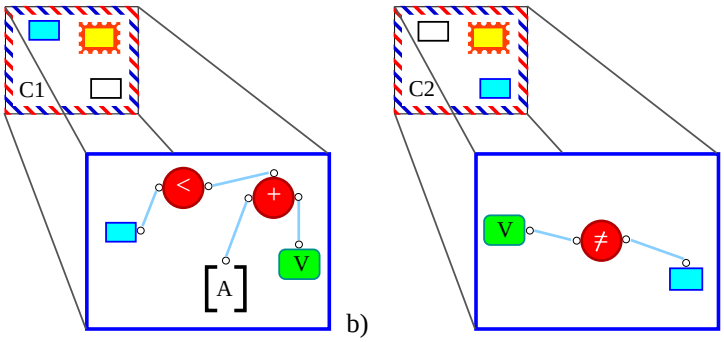
a) b)

Figura 3.47: Tipos de restricciones en objeto: a) Restricciones para recepción, b) Restricciones para delegación.

Consecuente con lo anterior hay dos tipos de íconos diferentes para las restricciones en los campos de recepción o delegación en un objeto:

1. Ícono para las restricciones de recepción: Se emplea un ícono en forma de carta, donde está resaltado con color el campo correspondiente al emisor del mensaje, este campo será llamado el Literal visual Remitente. Por ejemplo, en la Figura 3.48 a) al expandir la restricción C_1, el objeto que contiene esta restricción en el campo de recepción, sólo atenderá los mensajes enviados por los objetos que sean menores que la suma de la variable V y el argumento A.

2. Ícono para las restricciones de delegación: Se utiliza un ícono a manera de carta, el cual tiene resaltado con color el campo correspondiente al receptor del mensaje cuando éste no pueda ser contestado, en adelante éste campo será conocido como el Literal visual Destinatario. Por ejemplo, en la Figura 3.48 b), es notable que cuando el objeto que contenga esta restricción, no pueda contestar algún mensaje porque no contiene el método requerido, hará que lo envíe un nuevo objeto que sea diferente de la variable V.



a) b)

Figura 3.48: Ejemplos de restricciones en objeto: a) Ejemplo de una restricción para recepción, b) Ejemplo de una restricción para delegación.

- $RestriccionesRecepción \rightarrow$

  ($ListaRestriccionesRecepción \mid RestricciónRecepción$
- $RestoRestriccionesRecepción \rightarrow \wedge\ ListaRestriccionesRecepción \mid$ **)**
- $ListaRestriccionesRecepción \rightarrow$

  $RestricciónRecepción\ RestoRestriccionesRecepción$
- $RestricciónRecepción \rightarrow ParteFísica\ \sim\ Etiqueta\ :\ ParteLógicaR$
- $ParteLógicaR \rightarrow$

  $Literal\ Comparación\ Sig1 \mid Identificador\ Comparación\ Sig2$
- $Sig1 \rightarrow\ Identificador\ Resto1$
- $Sig2 \rightarrow\ Literal\ Resto1 \mid Identificador\ Resto2$
- $Resto1 \rightarrow\ Operación\ Identificador \mid \epsilon$
- $Resto2 \rightarrow\ Operación\ Literal \mid \epsilon$

- $RestriccionesDelegación = RestriccionesRecepción\{\textbf{forward / sender}\}$[a]

- $Valor \rightarrow ParteFísica\ \sim\ Número$
- $Par'ametro \rightarrow ParteFísica\ \sim\ NombreParámetro$

- $Literal \rightarrow ParteFísica\ \sim$ **sender**
- $Identificador \rightarrow ParteFísica\ \sim\ ParteLógicaIdentificador$
- $ParteLógicaIdentificador \rightarrow$

  $NombreVariable \mid Número \mid NombreParámetro$
- $SímboloComparación \rightarrow < \mid <= \mid > \mid >= \mid = \mid\ \#$
- $SímboloOperación \rightarrow +\ \mid\ -\ \mid\ *\ \mid\ /$

[a] Sustitución de **forward** por **sender** en todas las producciones que se generen en $RestriccionesRecepción$.

Figura 3.49: Sección de gramática para las restricciones de objeto.

- $Comparación \rightarrow ParteFísica \sim SímboloComparación$
- $Operador \rightarrow ParteFísica \sim SímboloOperación$

- $CuerpoObjeto \rightarrow ListaMétodos$
- $ListaMétodos \rightarrow Método\ RestoMétodos$
- $RestoMétodos \rightarrow \mathbf{\&}\ Método\ RestoMétodos \mid \epsilon$
  - $Método \rightarrow Nombre\ \mathbf{:}\ (\ ListaParámetros\ )\ Programa$
    - $ListaParámetros \rightarrow Parámetro\ RestoParámetros \mid \epsilon$
    - $RestoParámetros \rightarrow \mathbf{,}\ Parámetro\ RestoParámetros \mid \epsilon$

Figura 3.50: Sección de gramática para las restricciones de objeto (continuación).

Los procesos también pueden ser implicaciones que están compuestas de dos partes:

1. Antecedente: A su vez, puede ser de tres tipos:

   a) Imposición de restricciones: Es una lista de restricciones que, luego de su procesamiento, quedará almacenada hasta el fin de la ejecución del programa. Ejemplo en la Figura 3.51 a).

   b) Consulta de restricciones: compuesto por una lista de restricciones que son evaluadas para determinar el valor de verdad del antecedente, pero no son almacenadas. Ver la Figura 3.51 b).

   c) Envío de mensaje: se trata del atributo de un objeto pidiendo que sea enviado un mensaje. Éste último es referenciado por un ícono con sus respectivos Parámetros, como se muestra en la Figura 3.51 c)

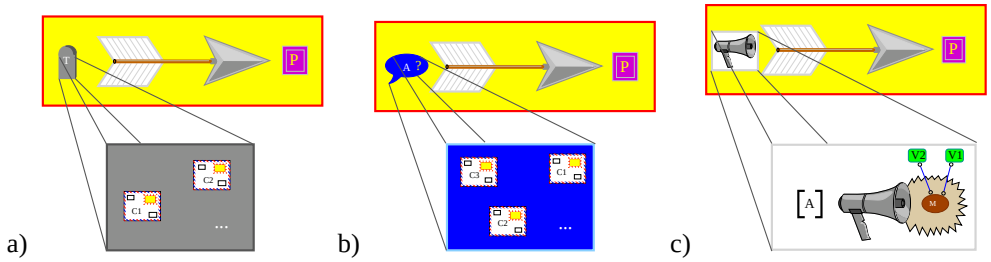


Figura 3.51: Tipos de antecedentes: a) Imposición de restricciones, b) Consulta de restricciones, c) Envío de mensaje.

Al analizar las expansiones de los antecedentes (ver Figura 3.51), las restricciones que hacen parte de una imposición o consulta difieren con respecto a las restricciones empleadas

en los campos delegación o recepción en un objeto, únicamente en la posibilidad de que no estén presentes los literales visuales de destinatario o remitente de un mensaje. Pues la única oportunidad en que una restricción de imposición o consulta puede emplear un literal visual (ya sea de Destinatario o Remitente) es cuando esta restricción se encuentra dentro de uno de los métodos de un objeto.

$ParteLógicaProceso \rightarrow$

viene de la Figura 3.46

$\vdots$

$\mid \quad \langle\ Antecedente\ ;\ Consecuente\ \rangle$

- $Antecedente \rightarrow ClaseRestricciones\ ListaRestricciones \mid Envío Mensaje$
  - $ListaRestricciones \rightarrow (\ Restricción\ RestoRestricciones \mid Restricción$
  - $RestoRestricciones \rightarrow \wedge\ Restricción\ RestoRestricciones \mid)$
  - $Restricción \rightarrow ParteFísica\ \sim\ Etiqueta\ :\ ParteLógicaRestricción$
  - $ParteLógicaRestricción \rightarrow Remitente\ Comparación\ Posterior1$
    $\mid Identificador\ Comparación\ Posterior2$
  - $Posterior1 \rightarrow Identificador\ Residuo1$
  - $Posterior2 \rightarrow Residuo1 \mid Identificador\ Residuo2$
  - $Residuo1 \rightarrow Operador\ Identificador \mid \epsilon$
  - $Residuo2 \rightarrow Operador\ Último \mid \epsilon$
  - $Último \rightarrow Identificador \mid Remitente$
    - $Remitente \rightarrow ParteFísica\ \sim\ \mathbf{sender}$
    - $ClaseRestricciones \rightarrow \mathbf{?} \mid \mathbf{!}$
  - $EnvíoMensaje \rightarrow Identificador \triangleleft\ Mensaje$
  - $Mensaje \rightarrow Nombre\ :\ [\ ListaAplicaciones\ ]$
    - $ListaAplicaciones \rightarrow Aplicación\ RestoAplicaciones \mid \epsilon$
    - $RestoAplicaciones \rightarrow ,\ Aplicación\ RestoAplicaciones \mid \epsilon$
      - $Aplicación \rightarrow (\ Puerto\ )\ Identificador$
        $Puerto \rightarrow ParteFísica\ \sim\ Nombrepuerto$

Figura 3.52: Sección de gramática para el proceso implicación y su componente antecedente.

2. Consecuente: Presentado en la Figura 3.53, está compuesto por un programa y éste, a su vez, conformado por un conjunto de procesos concurrentes. Esta parte denominada consecuente, sólo es procesada cuando las instrucciones del antecedente han tenido éxito; en otras palabras, cuando se han podido imponer las restricciones en el *Store* (almacén de restricciones), el resolutor ha contestado de manera afirmativa a la consulta de las restricciones o el mensaje ha sido enviado sin problemas.

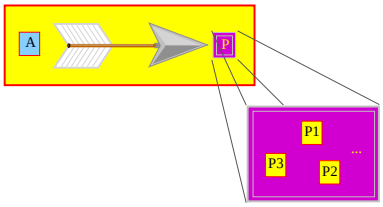


Figura 3.53: Consecuente.

- $Consecuente \rightarrow Programa$

Figura 3.54: Sección de gramática para el consecuente de una implicación

3.7. La relación de reducción de los constructores visuales GraPiCO

En esta sección se explicará la semántica operacional del cálculo GraPiCO por medio de la presentación de una relación de reducción con elementos visuales; antes, se introducirá la relación de reducción del cálculo PiCO presentado en [ADQ+01] por el grupo AVISPA, para una mejor comprensión.

3.7.1. Relación de reducción en el cálculo PiCO

El comportamiento de un proceso en cálculo PiCO es definido por transiciones desde una configuración inicial $\langle P; \top \rangle$. Una transición, $\langle P; S \rangle \longrightarrow$ [11] $\langle P'; S' \rangle$, significa que $\langle P; S \rangle$ puede ser transformada(reducida) en $\langle P'; S' \rangle$ por un paso simple en una computación.

COMM describe el resultado de la interacción entre un mensaje $I' \triangleleft l\,[\widetilde{K}]$ ***then*** Q y el objeto $(\phi_{\mathbf{sender}}, \delta_{\mathbf{forward}}) \triangleright [l : (\widetilde{x})\ P\ \&\ldots\ ]$. El $Store$[12] es utilizado para decidir si el objeto, en efecto, es el objetivo del mensaje.

[11] La relación de reducción, $\longrightarrow$, sobre configuraciones es la menor relación que satisface las reglas presetes en la Figura 3.55

[12] Repositorio de restricciones activas.

DEL describe el proceso delegación. Sea $I \triangleleft l\,[\widetilde{K}]\ \textbf{\textit{then}}\ Q$ un mensaje enviado al objeto $(\phi_{\mathbf{sender}}, \delta_{\mathbf{forward}}) \triangleright M$ y suponiendo que la etiqueta l no existe en M. En este caso, el mensaje es remitido a una nueva locación J que satisface la Condición de delegación.

Las reglas ASK y TELL describen la interacción entre los procesos de restricción y el $Store$.

PAR expresa que la reducción puede ocurrir antes de la composición. DEC-V es la manera de introducir nuevas variables y DEC-N es la forma de crear una localidad para un nombre de método.

COMM:
$$\frac{S \vdash_{\Delta} \phi[I'/\mathbf{sender}] \qquad |\widetilde{K}|=|\widetilde{x}|}{\left\langle I' \triangleleft l{:}[\widetilde{K}]\ \mathbf{then}\ Q \mid (\phi_{\mathbf{sender}},\delta_{\mathbf{forward}}) \triangleright [l{:}(\widetilde{x})P \& ...]; S \right\rangle \longrightarrow \left\langle Q \mid P\{\widetilde{K}/\widetilde{x}, I'/\mathbf{sender}\}; S \right\rangle}$$

DEL:
$$\frac{S \vdash_{\Delta} \phi[I'/\mathbf{sender}] \quad S \sqcup \delta[I'/\mathbf{forward}] \vdash_{\Delta} \bot \quad l \notin Labels(M)}{\left\langle \begin{pmatrix} I' \triangleleft l : [\widetilde{K}]\ \mathbf{then}\ Q \mid \\ (\phi_{\mathbf{sender}}, \delta_{\mathbf{forward}}) \triangleright M \end{pmatrix}; S \right\rangle \rightarrow \left\langle \begin{pmatrix} \mathbf{local}\ J\ \mathbf{in\ tell}\ \delta[J/\mathbf{forward}] \\ \mathbf{then}(J \triangleleft l : [\widetilde{K}]\ \mathbf{then}\ Q) \mid \\ (\phi_{\mathbf{sender}}, \delta_{\mathbf{forward}}) \triangleright M \end{pmatrix}; S \right\rangle}$$

TELL: $\langle \mathbf{tell}\ \phi\ \mathbf{then}\ P; S\rangle \longrightarrow \langle P; S \wedge \phi\rangle$

ASK: $\dfrac{S \vdash_{\Delta} \phi}{\langle \mathbf{ask}\ \phi\ \mathbf{then}\ P; S\rangle \longrightarrow \langle P; S\rangle}$, $\dfrac{S \vdash_{\Delta} \neg\phi}{\langle \mathbf{ask}\ \phi\ \mathbf{then}\ P; S\rangle \longrightarrow \langle O; S\rangle}$

PAR: $\dfrac{\langle P; S\rangle \longrightarrow \langle P'; S'\rangle}{\langle Q \mid P; S\rangle \longrightarrow \langle Q \mid P'; S'\rangle}$

DEC-V: $\dfrac{x \notin fv(S),\ \langle P; S \gg \{x\}\rangle \longrightarrow \langle P'; S'\rangle}{\langle \mathbf{local}\ x\ \mathbf{in}\ P; S\rangle \longrightarrow \langle P'; S'\rangle}$

DEC-N: $\dfrac{a \notin fn(S),\ \langle P; S \gg \{a\}\rangle \longrightarrow \langle P'; S'\rangle}{\langle \mathbf{local}\ a\ \mathbf{in}\ P; S\rangle \longrightarrow \langle P'; S'\rangle}$

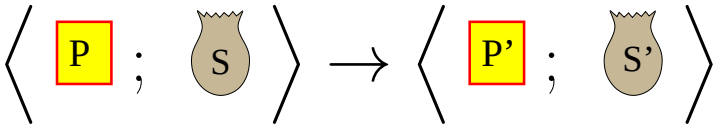
Figura 3.55: Sistema de transición.

3.7.2. Relación de reducción para los programas GraPiCO

De manera similar con el cálculo PiCO, el comportamiento de un programa GraPiCO es definido mediante transiciones donde, si una configuración A reduce a una B, significa que A puede ser transformada (reducida) en B mediante un paso simple de computación.

1. Relación de transición de procesos concurrentes:

   La regla presentada en la Figura 3.56 dice que un par de procesos concurrentes[13] Q y P con el $Store\ S$ puede ser reducido en los procesos concurrentes Q y P' con el $Store\ S'$, siempre que el proceso P con el $Store\ S$, en efecto, reduzca hacia el proceso P' y transforme el $Store$ hacia S'.



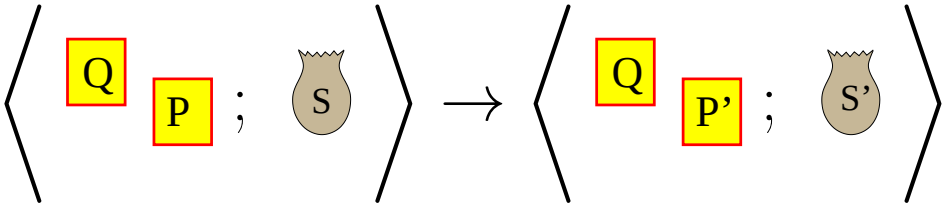


Figura 3.56: Reducción de concurrencia.

2. Relación de transición de comunicación entre los procesos objeto y envío de mensaje:

   La regla presentada en la Figura 3.57 plantea que los procesos concurrentes envío de mensaje y objeto tienen las siguientes características:

   - Envío de mensaje: Si el método M pedido por el Identificador I' (con la lista de argumentos $K1, \ldots, Kn$) es contestado, entonces es ejecutado el programa Q.
   - Objeto O: Cuenta como restricciones de recepción Gr con el conjunto de restricciones $C1, \ldots, Cm$ y dentro de sus métodos el etiquetado como M (con la lista de Parámetros $X1, \ldots, Xn$ y con cuerpo el programa P)

   Pueden ser reducidos a los programas concurrentes Q y P (donde se efectuan las sustituciones: del Identificador I' por el Literal visual receptor -etiquetado como s - y cada Parámetro Xi por su respectivo argumento Ki), siempre que el conjunto de restricciones Gr (con la sustitución en cada Ci del Literal visual receptor -etiquetado como s- por el Identificador $'I$) pueda ser deducido del $Store\ S$.

[13] Presentes en la misma ventana

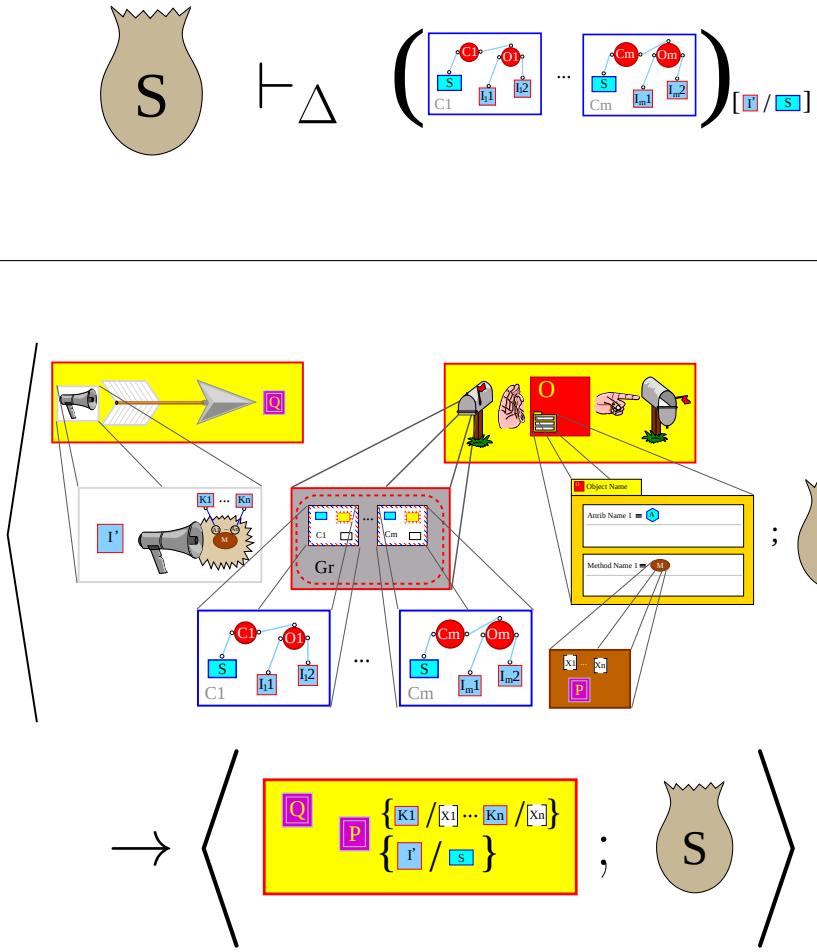


Figura 3.57: Reducción de comunicación.

3. Relación de transición del operador delegación en el proceso objeto:

   La regla presentada en la Figura 3.58 expone las siguientes características para los procesos concurrentes envío de mensaje y objeto:

   - Envío de mensaje: Si el método M pedido por el Identificador I' es contestado, entonces es ejecutado el programa C.
   - Objeto O: Tiene como restricciones de recepción al conjunto Gr_s, como restricciones de delegación a Gr_f y dentro de sus métodos NO está presente el etiquetado como M.

   Pueden ser reducidos a los procesos concurrentes con las siguientes características:

- Creación de nuevo ámbito de Identificador J en el programa $P1$: Donde $P1$ tiene como cuerpo un proceso implicación con antecedente la imposición del grupo de restricciones de recepción Gr_s del objeto O (con la sustitución del Literal visual de delegación etiquetado con f por el Identificador J), con consecuente el programa $P2$ que consiste en un proceso implicación, con antecedente el envío del mensaje M por parte del Identificador J, y como consecuente el programa C.
- Objeto O: Sin cambio alguno.

Siempre que el grupo de restricciones de recepción Gr_s (con la sustitución del Literal visual de recepción por el Identificador I' se pueda deducir del $Store\ S$ y que la unión de las restricciones impuestas en el $Store\ S$ y el grupo de restricciones de delegación Gr_f (con la sustitución del Literal visual de delegación f por el Identificador I') genere una contradicción.

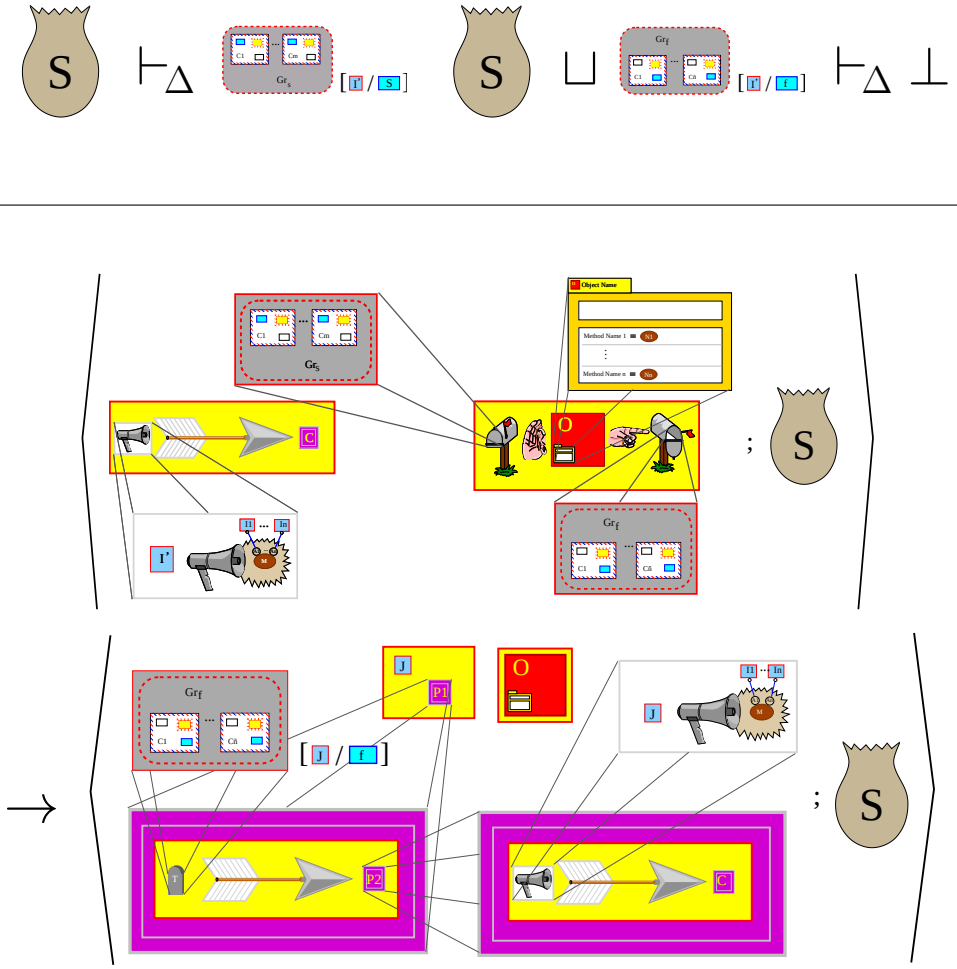


Figura 3.58: Reducción de delegación.

4. Relación de transición del proceso consulta de restricciones:

   La regla que se presenta en la Figura 3.59 muestra como los procesos implicación, con antecedente un proceso de consulta de restricciones con conjunto de restricciones Gr, y consecuente el programa P, pueden ser transformados en:

   - El programa P si del $Store\ S$ es deducible el conjunto de restricciones Gr.
   - El proceso vacío (inactivo o nulo) si la negación del conjunto de restricciones Gr es deducible del $Store\ S$.

   Al final, el $Store\ S$ queda intacto.

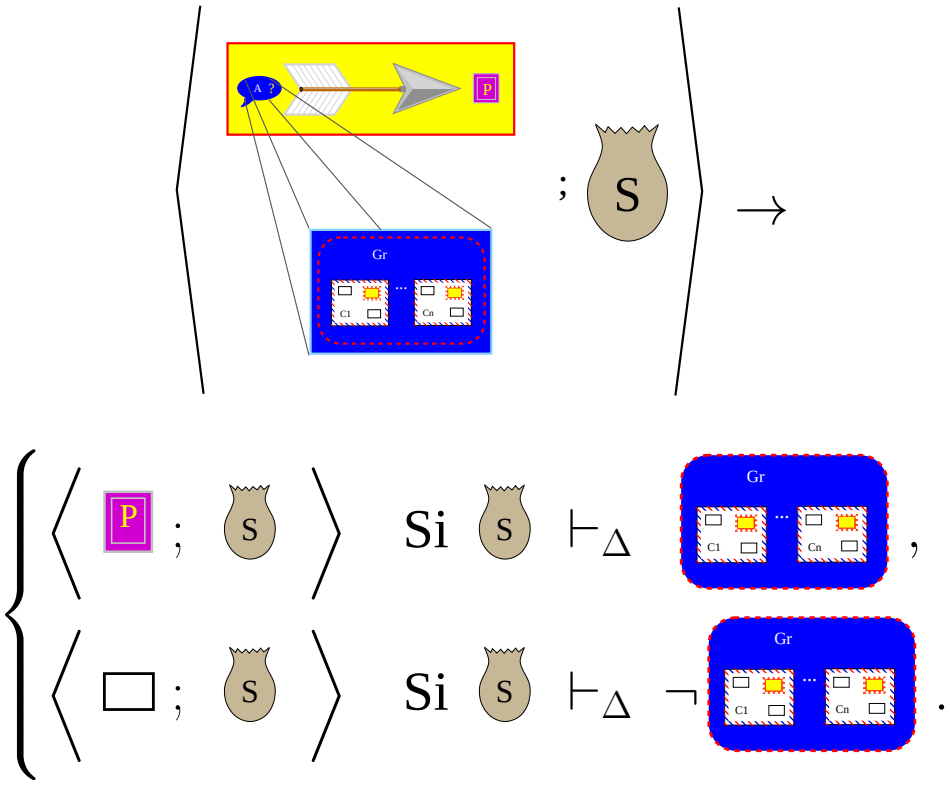


Figura 3.59: Reducción de una consulta de restricciones.

5. Relación de transición del proceso imposición de restricciones:

   La regla presentada en la Figura 3.60 expone que los procesos implicación con antecedente un proceso de imposición de restricciones con conjunto de restricciones Gr y consecuente el programa P, pueden ser transformados en el programa P acompañado del $Store\ S$ con la adición del grupo de restricciones Gr.

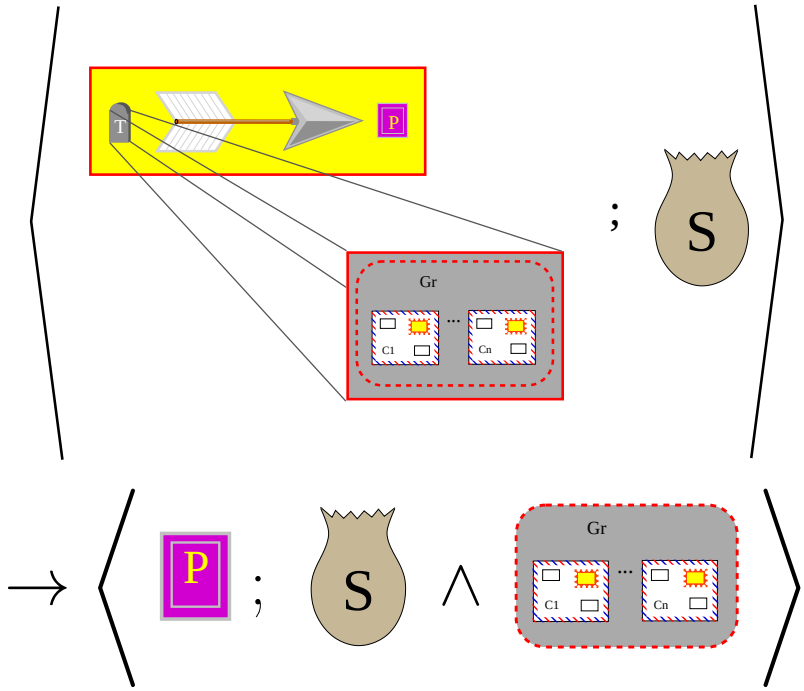


Figura 3.60: Reducción de imposición de restricciones.

6. Relación de transición del proceso creación de ámbito para una variable:

   La regla presentada en la Figura 3.61 presenta como un proceso de creación de ámbito de una variable V en un programa P se puede reducir al programa P' junto con el $Store\ S'$, siempre que P junto con el $Store\ S$ (adicionado con el equivalente visual de la restricción $V\ =\ V$) sean reducibles al programa P' junto con el $Store\ S'$.

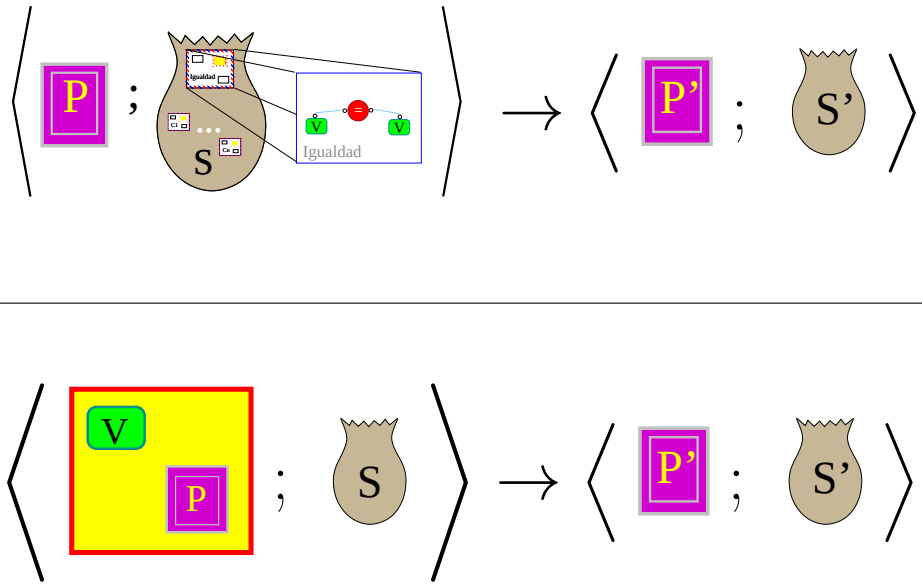


Figura 3.61: Reducción de creación de ámbito.

Capítulo 4

Reglas de traducción GraPiCO_Textual-cálculo PiCO

Luego de la presentación de la metodología de especificación textual de GraPiCO, en este capítulo se define y demuestra el mecanismo que permite obtener un programa PiCO a partir de un programa visual GraPiCO especificado en GraPiCO_Textual, y posteriormente se muestran las reglas de traducción construidas a partir de este mecanismo.

4.1. Introducción

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, se requiere guardar los programas visuales siguiendo las reglas de alguna especificación gramatical para mediante un mecanismo de traducción, obtener un código objeto.

La especificación mediante una Gsig, consiste en la representación textual de la información visual de un determinado constructor o conjunto de constructores. Empleando otros términos, una Gsig almacena en una expresión textual la información visual de cada constructor (referida como la parte física) y las relaciones con otros constructores del lenguaje (llamada la parte lógica). Dado que para la fase de traducción es indispensable la información contenida en la parte lógica, la traducción de un programa total o un constructor en particular dentro de un programa (especificado a través de Gsig) consiste en la separación de la parte lógica y su posterior traducción interna; de otro lado, la traducción de un grupo de constructores está compuesta por la traducción de cada uno de los constructores que hacen parte del conjunto.

4.2. Definición del mecanismo de traducción T_Gsig

Acorde con la definición de las Gsig en 5.2 la traducción (expresada mediante $\mathcal{T}$[1]) 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:

$$\begin{aligned}&\mathcal{T}[\![\ S'\ Sd_1\ \overbrace{(x-y)\ Path(X)}^{Parte\ Física}\ \sim\ Name(X)\ :\ ParteLógica\ Sd_2\ ]\!]\\ &= Ord(\mathcal{T}[\![\ S'\ ]\!],\ \mathcal{T}[\![\ Sd_1\ ]\!],\ \mathcal{T}[\![\ ParteLógica\ ]\!],\ \mathcal{T}[\![\ Sd_2\ ]\!])\end{aligned}$$

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

$$\begin{aligned}&\mathcal{T}[\![\ S'\ Sd_1\ X_1\ \widehat{Sc_1}\cdots\widehat{Sc_{n-1}}\ X_n\ Sd_2\ ]\!]\\ &= Ord(\mathcal{T}[\![S']\!],\ \mathcal{T}[\![Sd_1]\!],\ \mathcal{T}[\![X_1]\!],\ \ \mathcal{T}[\![\widehat{Sc_1}]\!],\ \cdots\ \mathcal{T}[\![\widehat{Sc_{n-1}}]\!],\ \ \mathcal{T}[\![X_n]\!],\ \mathcal{T}[\![Sd_2]\!])\end{aligned}$$

Del caso 1. podemos obtener, que la traducción de un constructor simple generado por una Gsig (código fuente) hacia su respectivo código de lenguaje textual (código objeto), consiste en:

1. La supresión de la parte física del código fuente.
2. La traducción de la parte lógica del código fuente.
3. La transformación de los símbolos de sincronización del código fuente, en los correspondientes símbolos de sincronización del código objeto.
4. Organización de las anteriores traducciones mediante una función ($Ord()$).

Lo anterior es consecuente con la estructura de las gramáticas de sistemas de íconos generalizados Gsig, dado que en la parte física es donde queda almacenada la información que relaciona los constructores desde el punto de vista netamente visual y la parte lógica, es donde está la definición de la estructura interna.

Entonces, se define la traducción de cada constructor generado por una Gsig con la estructura presentada en la ecuación 4.1 mediante un objeto con el diseño presentado en la Figura 4.1 (Donde **Type** se refiere al tipo de datos elegido al momento de desarrollar, para almacenar la especificación mediante una Gsig).

$$\underbrace{\langle X\rangle}_{\text{No terminal representado } X}\ \rightarrow \overbrace{S'\ Sd_1\ Y\ \sim\ Etiqueta\ :\ \underbrace{Z}_{\text{Parte lógica de } X\ \stackrel{\text{def}}{=} z\_Gsig}\ Sd_2}^{\text{Representación Gsig de } X\ \stackrel{\text{def}}{=} x\_Gsig} \qquad (4.1)$$

[1]En adelante se usará $\mathcal{T}[\![\ Y\ ]\!]$ como la función semántica que entrega la traducción en código de un lenguaje textual, de una expresión Y, generada por una Gsig .

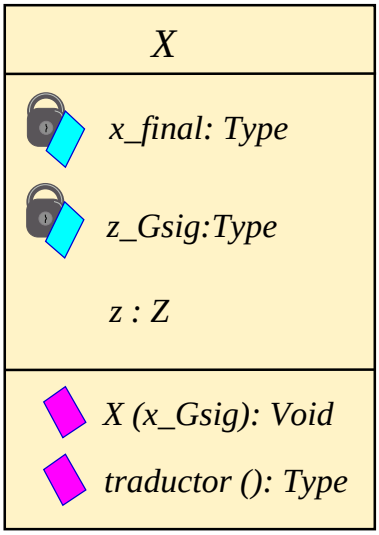


Figura 4.1: Diseño de objeto del constructor simple X.

Posterior al diseño del objeto del constructor simple X, se presenta en las Figuras 4.2 y 4.3 una plantilla para la implementación de la traducción de los constructores visuales simples especificados por una Gsig.

```
//Clase que efectúa la traducción del constructor visual simple X.
public class X{
        //Código traducido del constructor X.
        private Type x_final;

        //Parte lógica de X. En otras palabras, Z en términos de Gsig.
        private Type z_Gsig ;

        /*Definición del objeto z,
        como instancia de la clase de la parte lógica de X, denominada Z.*/
        Z z;
```

Figura 4.2: Implementación de objeto del constructor simple X, sección: atributos.

/*Constructor de las instancias de X, que recibe al constructor X en Gsig, luego 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', Sd_1, Sd_2 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 **=**

$ordenación$ **(**$\mathcal{T}[\![\ S'\ ]\!]$, $\mathcal{T}[\![\ Sd_1\ ]\!]$, $z.traductor$**()**, $\mathcal{T}[\![\ Sd_2\ ]\!]$**);**

return x_final**;**

}

}

Figura 4.3: Implementación de objeto del constructor simple X, sección: métodos.

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 [2] se efectúa mediante la extracción de segmentos de código delimitado por símbolos de sincronización.

De la plantilla de las Figuras 4.2 y 4.3 se aprecia que la traducción de un constructor visual 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 que está delimitada por los símbolos de sincronización **:** y Sd_2 y su posterior almacenamiento en z_Gsig.

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

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

X) y, por ende, se activa el constructor de z y su respectivo método $traductor$. De esta manera, sucesivamente se navega el árbol sintáctico de manera virtual, por medio de la jerarquía de clases.

Finalmente, cuando las diferentes partes lógicas ya están traducidas, se procede a su disposición con la función $ordenación$, a fin de obtener un código que se ajuste a las reglas sintácticas del lenguaje textual objeto. El diseño de las funciones $separador$ y $ordenación$ y el método $traductor$ dependen del lenguaje que se esté utilizando para realizar la implementación.

De otro lado, por el caso 2, está que la traducción de un constructor visual compuesto especificado textualmente utilizando una Gsig, consiste en:

1. Traducción de cada constructor X_i que compone el constructor.
2. Traducción del símbolo de identificación única S'.
3. Transformación de los símbolos de delimitación Sd_1 y Sd_2.
4. Transformación de cada símbolo de sincronización $\widehat{Sc_i}$.
5. Organización de todas las anteriores traducciones mediante una función ($Ord()$).

Dadas las observaciones sobre el mecanismo de traducción, se define la traducción de cada constructor visual compuesto generado por una Gsig con la estructura presentada en la ecuación 4.2 y mediante un objeto con el diseño mostrado en la Figura 4.4.

$$L_X \rightarrow \overbrace{S'\ Sd_1 \quad \underbrace{X_1}_{\text{Representación Gsig de } X_1 \stackrel{\text{def}}{=} x_1\_Gsig} \quad \widehat{Sc_1} \cdots \widehat{Sc_{n-1}}\ X_n\ Sd_2}^{\text{Representación Gsig de } L_X \stackrel{\text{def}}{=} lx\_Gsig} \tag{4.2}$$

Después del diseño del objeto del constructor visual compuesto L_X, se muestra en las Figuras 4.5 y 4.6 una plantilla para la implementación de la traducción de un constructor visual compuesto especificado por una Gsig.

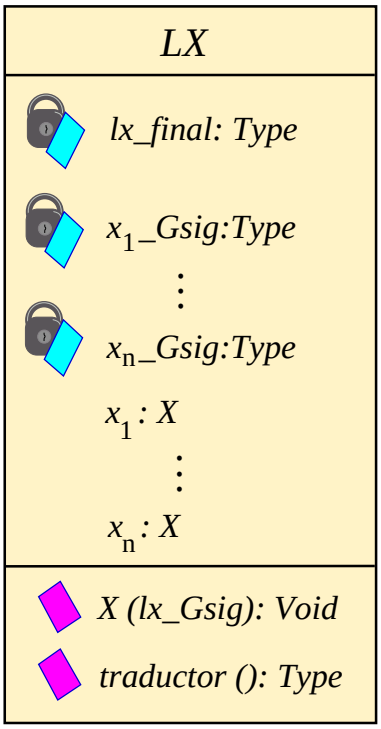


Figura 4.4: Diseño de objeto del constructor visual compuesto L_X.

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

        //Constructores de la lista en términos de Gsig.
        private Type x_1_Gsig, ··· , x_n_Gsig;

        /*Definición de cada objeto x_i como instancia de la clase
        del constructor X_i respectivo.*/
        X_1 x_1; ··· ; X_n x_n;
```

Figura 4.5: Implementación de objeto para la traducción del constructor visual compuesto L_X.

/*Constructor de la instancia de L_X que recibe el constructor visual compuesto L_X en Gsig después se encarga de extraer el código de cada constructor X_i por medio de $separador_i$ y lo almacena en x_i_Gsig.*/

public L_X **(*Type*** lx_Gsig**) {**

$x_1\_Gsig \;=\; separador_1$**(**$lx\_Gsig$**);**

$\vdots$

$x_n\_Gsig \;=\; separador_n$**(**$lx\_Gsig$**);**

}

//Método que se encarga de traducir el constructor L_X.

public Type $traductor$ **() {**

/*Creación del objeto x_i,
a partir de x_i_Gsig.*/

x_1 **=** ***new*** X_1**(**x_1_Gsig**);**

$\vdots$

x_n **=** ***new*** X_n**(**x_n_Gsig**);**

/*Obtención del código de L_X mediante el ordenamiento de la traducción de sus componentes (constructores y símbolos)*/

$lx\_final \;=$

$ordenación$ ($\mathcal{T}[\![Sd_1]\!]$, $x_1.traductor()$,
$\mathcal{T}[\![\widehat{Sc_1}]\!]$, $\cdots$, $\mathcal{T}[\![\widehat{Sc_{n-1}}]\!]$,
$x_n.traductor()$, $\mathcal{T}[\![Sd_2]\!]$);

return x_final**;**

}

}

Figura 4.6: Implementación de objeto para traducción del constructor visual compuesto L_X.

De igual forma que con la plantilla de las Figuras 4.2 y 4.3, con la plantilla presentada en las Figuras 4.5 y 4.6 es notable que la traducción de un constructor visual compuesto L_X inicia con la creación del objeto lx (instancia de la clase L_X), 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, que están delimitados por sendos símbolos de sincronización $\widehat{Sc_{i-1}}$ y $\widehat{Sc_i}$ (cuando $i \neq 1, n$) , Sd_1 y $\widehat{Sc_1}$ (cuando $i = 1$) , o $\widehat{Sc_{n-1}}$ y Sd_2 (cuando $i = n$), y su posterior almacenamiento en x_i_Gsig respectivamente. Luego, con la activación de ca-

da método $traductor_i$, con la información contenida en cada x_i_Gsig se crea el objeto x_i (instancia de la clase correspondiente al componente X_i) y, por ende, se activa el constructor de cada x_i. Al final, cuando los diferentes componentes ya están traducidos se procede a su arreglo con la función $ordenación$, a fin de obtener un código que se ajuste a las normas sintácticas del lenguaje objeto.

4.3. Prueba de la *corrección* de la función de traducción $\mathcal{T}$

Para proceder con la demostración de la *corrección* primero se presenta la prueba de la *validez* y posteriormente la de la *completitud*.

4.3.1. Demostración de la *validez* de la función de traducción $\mathcal{T}$

Planteamiento recursivo de la función $\mathcal{T}$

La función $\mathcal{T}$ para $\mathcal{E}[\![X]\!]$ (ver Figura 4.7) presentada en la fórmula 4.3 se calcula sobre la 4-upla en 4.4.

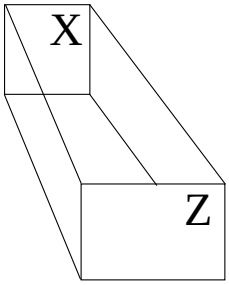


Figura 4.7: Expansión del constructor X en el constructor Z.

$$\mathcal{T}[\![\mathcal{E}[\![X]\!]]\!] = \mathcal{T}[\![S'_X\ {Sd_1}_X\ PF_X \sim\ Name(X)\ :\ PL_X\ {Sd_2}_X]\!] = x\_final \quad (4.3)$$

La parte lógica del constructor X consiste en $\mathcal{E}[\![Z]\!]$ que, de ahora en adelante, será conocida como Y.

$$\langle \mathcal{T}[\![S'_X]\!], \mathcal{T}[\![{Sd_1}_X]\!], \mathcal{T}[\![Y]\!], \mathcal{T}[\![{Sd_2}_X]\!] \rangle \quad (4.4)$$

Donde la parte lógica de X corresponde a 4.5

$$Y\ =\ S'_Z\ {Sd_1}_Z\ PF_Z \sim\ Name(Z)\ :\ PL_Z\ {Sd_2}_Z \quad (4.5)$$

De igual forma, la función $\mathcal{T}$ para $\mathcal{E}[\![Z]\!]$ presentada en la fórmula 4.6 se calcula sobre la 4-upla presentada en 4.7.

$$\mathcal{T}[\![S'_Z\ Sd1_Z\ PF_Z \sim\ Name(Z)\ :\ PL_Z\ Sd2_Z]\!] = z\_final \tag{4.6}$$

$$\langle \mathcal{T}[\![S'_Z]\!], \mathcal{T}[\![Sd1_Z]\!], \mathcal{T}[\![PL_Z]\!], \mathcal{T}[\![Sd2_Z]\!] \rangle \tag{4.7}$$

Luego, el planteamiento recursivo de la función $\mathcal{T}$ se expresa mediante el diagrama conmutativo de la Figura 4.8:

$$\begin{array}{ccc} \mathcal{T}\left[\!\left[\mathcal{E}[\![X]\!]\right]\!\right] & \xrightarrow{sobre} & \langle \mathcal{T}[\![S'_X]\!], \mathcal{T}[\![Sd1_X]\!], \mathcal{T}[\![Y]\!], \mathcal{T}[\![Sd2_X]\!] \rangle \ ^{a)} \\ \downarrow & & \downarrow \\ \mathcal{T}\left[\!\left[\mathcal{E}[\![X]\!]\right]\!\right] & \xrightarrow{sobre} & \langle \mathcal{T}[\![S'_X]\!], \mathcal{T}[\![Sd1_X]\!], z\_final, \mathcal{T}[\![Sd2_X]\!] \rangle \ ^{b)} \end{array}$$

Dado que en b) z_final ya está calculado y como a) y b) son del mismo tipo, se afirma que b) es más $sencillo$ que a).

Figura 4.8: Diagrama conmutativo del planteamiento recursivo de la función de traducción $\mathcal{T}$.

Definición 9. *Si C es un segmento de código, entonces cualquier $aserción$[3] $\{P\}$ se denomina $precondición$ de C si $\{P\}$ sólo implica el estado inicial. Cualquier $aserción$ $\{Q\}$ se denomina $poscondición$ si $\{Q\}$ únicamente implica el estado final. Si C tiene como precondición a $\{P\}$ y como poscondición a $\{Q\}$, se escribe $\{P\}\ C\ \{Q\}$. La terna $\{P\}\ C\ \{Q\}$ se denomina* **terna de Hoare**.

Definición 10. *Seguirá denominándose con ℓ(G) al conjunto de todas las posibles palabras generadas con la gramática G; es decir, el lenguaje de G.*

En consecuencia, la función $\mathcal{T}$ puede ser especificada por la terna de Hoare en la Figura 4.9.

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

Figura 4.9: Especificación de la función traducción $\mathcal{T}$.

Dado lo anterior, verificar la *validez* de $\mathcal{T}$ equivale a demostrar la veracidad del predicado en 4.8.

$$\forall_{X \in\, \ell(\text{ GraPiCO })}.\ \mathcal{E}[\![X]\!] \in \ell(\text{ GraPiCO\_Textual }) \rightarrow \mathcal{T}\left[\!\left[\mathcal{E}[\![X]\!]\right]\!\right] \in \ell(\text{ PiCO }) \tag{4.8}$$

[3]Se denomina $aserción$ a cualquier sentencia referente a un estado de programa.

Entonces, es necesario algún tipo de inducción sobre el conjunto definido en 4.9.

$$\text{GraPiCO\_}\mathcal{E} = \left\{ X \in \ell(\text{ GraPiCO }) : \mathcal{E}[\![X]\!] \in \ell(\text{ GraPiCO\_Textual }) \right\} \tag{4.9}$$

Relación binaria $\succ$ en ℓ(GraPiCO)

Sea la relación binaria $\succ$ en ℓ(GraPiCO), $\succ \subset \ell$(GraPiCO) $\times$ ℓ(GraPiCO) definida en 4.11 (primero se presenta la definición del conjunto cerradura de la función de expansión en 4.10).

$$Ex(X)^* = \left\{ Y : Y \in Ex(X) \vee Z \in Ex(X) \rightarrow Y \in Ex(Z)^* \right\} \tag{4.10}$$

$$X \succ Y \stackrel{\text{def}}{=} Y \in Ex(X)^* \tag{4.11}$$

Después se verifica la transitividad de la relación $\succ$ en 4.12.

$$(T) \bigvee_{X,Y,Z \in \ell(\text{ GraPiCO })} \cdot \frac{\begin{array}{c} X \succ Y \rightarrow Y \in Ex(X)^* \\ Y \succ Z \rightarrow Z \in Ex(Y)^* \end{array}}{Z \in Ex(X)^* \rightarrow X \succ Z} \tag{4.12}$$

Lista de reglas de traducción de GraPiCO_Textual en cálculo PiCO

Para mostrar la traducción de un programa GraPiCO_Textual hacia PiCO se emplea una lista de fórmulas de igualdad, 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.

1. Traducción de un programa:

$$\begin{aligned} \mathcal{T}[\![Programa]\!] &\equiv \mathcal{T}[\![\, \{\, ParteFísica \sim Etiqueta : ParteLógicaPrograma \,\}\,]\!] \\ &\equiv \mathcal{T}[\![\underbrace{\epsilon}_{S'}]\!]\, \mathcal{T}[\![\underbrace{\{}_{Sd_1}]\!]\, \mathcal{T}[\![\, ParteLógicaPrograma \,]\!]\, \mathcal{T}[\![\underbrace{\}}_{Sd_1}]\!] \end{aligned} \tag{4.13}$$

   - Traducción de los símbolos de sincronización: $\mathcal{T}[\![\, \{ \,]\!] \equiv \epsilon,\ \mathcal{T}[\![\, \} \,]\!] \equiv$.

$$\mathcal{T}[\![Programa]\!] \equiv \mathcal{T}[\![\, ParteLógicaPrograma \,]\!] \text{ .}$$

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.

$$
\begin{aligned}
a)\quad & \mathcal{T}[\![ParteLógicaPrograma]\!] \\
& \equiv \mathcal{T}[\![\ (\ Proceso_1\ \mid\ \ldots\ \mid\ Proceso_n\ )\ ]\!] \\
& \equiv \mathcal{T}[\![\ \underbrace{\epsilon}_{S'}\ ]\!]\,\mathcal{T}[\![\ \underbrace{(}_{Sd_1}\ ]\!]\,\mathcal{T}[\![Proc._1]\!]\,\mathcal{T}[\![\ \underbrace{\mid}_{\widehat{Sc_1}}\ ]\!]\ \ldots\ \mathcal{T}[\![\ \underbrace{\mid}_{\widehat{Sc_{n-1}}}\ ]\!]\,\mathcal{T}[\![Proc._n]\!]\,\mathcal{T}[\![\ \underbrace{)}_{Sd_2}\ ]\!] \\
& \equiv (\ \mathcal{T}[\![Proceso_1]\!]\ \mid\ \ldots\ \mid\ \mathcal{T}[\![Proceso_n]\!]\ ) \\
\\
b)\quad & \mathcal{T}[\![ParteLógicaPrograma]\!] \ \equiv\ \mathcal{T}[\![\ Proceso\ ]\!]
\end{aligned}
\tag{4.14}
$$

3. Traducción de un proceso:

$$
\begin{aligned}
\mathcal{T}[\![\textsf{Proceso}]\!] &\equiv \mathcal{T}[\![\ Clonación\ ParteFísica\ \sim\ Etiqueta : ParteLógicaProceso\ ]\!] \\
&\equiv \mathcal{T}[\![\ \underbrace{Clonación}_{S'}\ ]\!]\,\mathcal{T}[\![\ \underbrace{\epsilon}_{Sd_1}\ ]\!]\,\mathcal{T}[\![\ ParteLógicaProceso\ ]\!]\,\mathcal{T}[\![\ \underbrace{\epsilon}_{Sd_2}\ ]\!]
\end{aligned}
\tag{4.15}
$$

   - Traducción de la Condición de replicación:

$$
\mathcal{T}[\![clonación]\!] = \begin{cases} \textbf{\textit{clone}} & \text{Si } clonación \equiv *, \\ \epsilon & \text{Si } clonación \equiv \epsilon. \end{cases}
\tag{4.16}
$$

4. Traducción de los casos de los que se puede tratar la parte lógica de un proceso:

   Cuando la parte lógica de un proceso corresponde a:

   - Definición de ámbito:

$$
\begin{aligned}
& \mathcal{T}[\![ParteLógicaProceso]\!] \\
\equiv\ & \mathcal{T}[\![\ [\ ListaVariables\ ;\ ListaNombres\ ;\ Programa\ ]\ ]\!] \\
\equiv\ & \mathcal{T}[\![\ \underbrace{\epsilon}_{S'}\ ]\!]\,\mathcal{T}[\![\ \underbrace{[}_{Sd_1}\ ]\!]\,\mathcal{T}[\![\ ListaVariables\ ]\!]\,\mathcal{T}[\![\ \underbrace{;}_{\widehat{Sc_1}}\ ]\!] \\
& \qquad \mathcal{T}[\![\ ListaNombres\ ]\!]\,\mathcal{T}[\![\ \underbrace{;}_{\widehat{Sc_2}}\ ]\!]\,\mathcal{T}[\![\ Programa\ ]\!]\,\mathcal{T}[\![\ \underbrace{]}_{Sd_2}\ ]\!]
\end{aligned}
\tag{4.17}
$$

   - Traducción de los símbolos de sincronización: $\mathcal{T}[\![\ [\ ]\!] \equiv \textbf{\textit{local}}$, $\mathcal{T}[\![\ ]\ ]\!] \equiv \epsilon$, $\mathcal{T}[\![\ \underbrace{;}_{\widehat{Sc_2}}\ ]\!] \equiv \textbf{\textit{in}}$

$$\mathcal{T}[\![\underbrace{\ ;\ }_{\widehat{Sc_1}}]\!] = \begin{cases}, & \text{Si } {G_v}^a \neq \emptyset \wedge {G_n}^b \neq \emptyset, \\ \epsilon & \text{en otro caso.}\end{cases} \tag{4.18}$$

[a] Grupo de variables
[b] Grupo de nombres

$$\begin{aligned}&\mathcal{T}[\![ParteLógicaProceso]\!] \\ \equiv\ &\textbf{\textit{local}}\mathcal{T}[\![\ ListaVariables\ ]\!]\mathcal{T}[\![\underbrace{\ ;\ }_{\widehat{Sc_1}}]\!]\ \mathcal{T}[\![\ ListaNombres\ ]\!]\textbf{\textit{in}}\mathcal{T}[\![Programa]\!]\end{aligned}$$

- Traducción de lista de Identificadores (variables, Parámetros o nombres):

$$\begin{aligned}&\mathcal{T}[\![ListaIdentificadores]\!] \\ \equiv\ &\mathcal{T}[\![Identificador_1\ ,\ \ldots\ ,\ Identificador_n]\!] \\ \equiv\ &\mathcal{T}[\![\underbrace{\epsilon}_{S'}]\!]\ \mathcal{T}[\![\underbrace{\epsilon}_{Sd_1}]\!]\ \mathcal{T}[\![Identificador_1]\!]\ \mathcal{T}[\![\underbrace{\ ,\ }_{\widehat{Sc_1}}]\!] \\ &\qquad\qquad \ldots\ \mathcal{T}[\![\underbrace{\ ,\ }_{\widehat{Sc_{n-1}}}]\!]\ \mathcal{T}[\![Identificador_n]\!]\ \mathcal{T}[\![\underbrace{\epsilon}_{Sd_2}]\!] \\ \equiv\ &\mathcal{T}[\![Identificador_1]\!]\ ,\ldots\ ,\ \mathcal{T}[\![Identificador_n]\!]\end{aligned} \tag{4.19}$$

  - Traducción de un Identificador (variable, Parámetro, nombre o Literal de recepción o delegación):

$$\begin{aligned}&\mathcal{T}[\![Identificador]\!] \\ \equiv\ &\mathcal{T}[\![\underbrace{\epsilon}_{S'}]\!]\ \mathcal{T}[\![\underbrace{\epsilon}_{Sd_1}]\!]\ \mathcal{T}[\![\ ParteFísica\ \sim\ NomIdent.\ ]\!]\ \mathcal{T}[\![\underbrace{\epsilon}_{Sd_2}]\!] \\ \equiv\ &NomIdentificador\end{aligned} \tag{4.20}$$

- Implicación:

$$\begin{aligned}&\mathcal{T}[\![ParteLógicaProceso]\!] \\ \equiv\ &\mathcal{T}[\![\ \langle\ Antecedente\ ;\ Consecuente\ \rangle\ ]\!] \\ \equiv\ &\mathcal{T}[\![\underbrace{\epsilon}_{S'}]\!]\ \mathcal{T}[\![\underbrace{\langle}_{Sd_2}]\!]\ \mathcal{T}[\![Antecedente]\!]\ \mathcal{T}[\![\underbrace{\ ;\ }_{\widehat{Sc_1}}]\!]\mathcal{T}[\![Consecuente]\!]\mathcal{T}[\![\underbrace{\rangle}_{Sd_2}]\!]\end{aligned} \tag{4.21}$$

  - Traducción de los símbolos de sincronización: $\mathcal{T}[\![\ \langle\ ]\!]$, $\mathcal{T}[\![\ \rangle\ ]\!] \equiv \epsilon$ y $\mathcal{T}[\![\ ;\ ]\!] \equiv$ ***then***

$$\begin{aligned} & \mathcal{T}[\![ParteLógicaProceso]\!] \\ \equiv\ & \mathcal{T}[\![Antecedente]\!] \ \textbf{\textit{then}}\ [\![Consecuente]\!] \end{aligned}$$

Traducción de los posibles casos en los que puede consistir un antecedente en un proceso de implicación:

$$\mathcal{T}[\![Antecedente]\!] \equiv \mathcal{T}[\![Restricciones]\!] \ \vee\ \mathcal{T}[\![EnvioMensaje]\!] \tag{4.22}$$

- Traducción de un antecedente (restricciones) cuando contenga una lista de restricciones:

$$\begin{aligned} & \mathcal{T}[\![Restricciones]\!] \\ \equiv\ & \mathcal{T}[\![ClaseRestricciones\ ListaRestricciones\ ]\!] \\ \equiv\ & \mathcal{T}[\![\ \underbrace{ClaseRestricciones}_{S'}\ ]\!]\ \mathcal{T}[\![ListaRestricciones]\!] \end{aligned} \tag{4.23}$$

  - Traducción de clases de restricciones:

$$\mathcal{T}[\![ClaseRestricciones]\!] = \begin{cases} \textbf{\textit{ask}} & Si\ ClaseRestricciones\ =\ \textbf{?}, \\ \textbf{\textit{tell}} & Si\ ClaseRestricciones\ =\ \textbf{!}. \end{cases} \tag{4.24}$$

  - Traducción de casos de listas de restricciones:
    a) Lista compuesta de varias restricciones, b) Lista compuesta de una única restricción:

$$\begin{aligned} a)\quad & \mathcal{T}[\![\ ListaRestricciones\ ]\!] \\ & \equiv \mathcal{T}[\![\ (\ Restricción_1\ \wedge\ \ldots\ \wedge\ Restricción_n\ )\ ]\!] \\ & \equiv \mathcal{T}[\![\ \underbrace{(}_{Sd_1}\ ]\!]\mathcal{T}[\![Rest._1]\!]\mathcal{T}[\![\ \underbrace{\wedge}_{\widehat{Sc_1}}\ ]\!]\ldots\mathcal{T}[\![\ \underbrace{\wedge}_{\widehat{Sc_{n-1}}}\ ]\!]\mathcal{T}[\![Rest._n]\!]\mathcal{T}[\![\ \underbrace{)}_{Sd_1}\ ]\!] \\ & \equiv (\ \mathcal{T}[\![\ Restricción_1\ ]\!]\ \wedge\ \ldots\ \wedge\ \mathcal{T}[\![\ Restricción_n\ ]\!]\ ) \end{aligned}$$

$$b)\quad \mathcal{T}[\![\ ListaRestricciones\ ]\!] \ \equiv\ \mathcal{T}[\![\ Restricción\ ]\!] \tag{4.25}$$

    - Traducción de los casos de una restricción:
      a) Restricción compuesta por código en tres direcciones.
      b) Restricción compuesta por código en dos direcciones.

$$
\begin{aligned}
a)\quad & \mathcal{T}[\![\ Restricción\ ]\!] \\
& \equiv \mathcal{T}[\![\ I_1\ O\ I_2\ C\ I_3\ ]\!] \\
& \equiv \mathcal{T}[\![\ \underbrace{\epsilon}_{S'}\ ]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{Sd_1}\ ]\!]\ \mathcal{T}[\![\ I_1\ ]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{\widehat{Sc_1}}\ ]\!]\ \mathcal{T}[\![\ C\ ]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{\widehat{Sc_2}}\ ]\!] \\
& \qquad \mathcal{T}[\![\ I_2\ ]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{\widehat{Sc_3}}\ ]\!]\ \mathcal{T}[\![\ O\ ]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{\widehat{Sc_4}}\ ]\!]\ \mathcal{T}[\![\ I_3\ ]\!]\mathcal{T}[\![\ \underbrace{\epsilon}_{Sd_2}\ ]\!] \\
& \equiv \mathcal{T}[\![\ I_1\ ]\!]\ \mathcal{T}[\![\ C\ ]\!]\ \mathcal{T}[\![\ I_2\ ]\!]\ \mathcal{T}[\![\ O\ ]\!]\ \mathcal{T}[\![\ I_3\ ]\!] \\
\\
b)\quad & \mathcal{T}[\![\ Restricción\ ]\!] \\
& \equiv \mathcal{T}[\![\ I_1\ C\ I_2\ ]\!] \\
& \equiv \mathcal{T}[\![\ \underbrace{\epsilon}_{S'}\ ]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{Sd_1}\ ]\!]\ \mathcal{T}[\![\ I_1\ ]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{\widehat{Sc_1}}\ ]\!]\ \mathcal{T}[\![\ C\ ]\!] \\
& \qquad \mathcal{T}[\![\ \underbrace{\epsilon}_{\widehat{Sc_2}}\ ]\!]\ \mathcal{T}[\![\ I_2\ ]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{Sd_2}\ ]\!] \\
& \equiv \mathcal{T}[\![\ I_1\ ]\!]\ \mathcal{T}[\![\ C\ ]\!]\ \mathcal{T}[\![\ I_2\ ]\!]
\end{aligned}
\tag{4.26}
$$

- Traducción de un antecedente y sus componentes (mensaje, listaplicaciones y aplicación) cuando se trata de un envío de mensaje:

$$
\begin{aligned}
& \mathcal{T}[\![EnvioMensaje]\!] \\
& \equiv \mathcal{T}[\![\ Identificador\ \lhd\ Mensaje\ ]\!] \\
& \equiv \mathcal{T}[\![\ \underbrace{\epsilon}_{S'}\ ]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{Sd_1}\ ]\!]\ \mathcal{T}[\![\ Ident.\ ]\!]\ \mathcal{T}[\![\ \underbrace{\lhd}_{\widehat{Sc_1}}\ ]\!]\ \mathcal{T}[\![Mensaje]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{Sd_2}\ ]\!] \\
& \equiv \mathcal{T}[\![\ Identificador\ ]\!]\ \lhd\ \mathcal{T}[\![Mensaje]\!]
\end{aligned}
\tag{4.27}
$$

  - Traducción de mensaje dentro de un envío de mensaje:

$$
\begin{aligned}
& \mathcal{T}[\![Mensaje]\!] \\
& \equiv \mathcal{T}[\![\ NombreMensaje\ \texttt{:}\ [\ ListaAplicaciones\ ]\ ]\!] \\
& \equiv \mathcal{T}[\![\ \underbrace{NombreMensaje\ \texttt{:}}_{S'}\ ]\!]\ \mathcal{T}[\![\ \underbrace{[}_{Sd_1}\ ]\!]\ \mathcal{T}[\![\ ListaApl.\ ]\!]\ \mathcal{T}[\![\ \underbrace{]}_{Sd_2}\ ]\!] \\
& \equiv NombreMensaje\ \texttt{:}\ [\ \mathcal{T}[\![ListaAplicaciones]\!]\ ]
\end{aligned}
\tag{4.28}
$$

    - Traducción de una lista de Aplicaciones dentro de un mensaje:

$$
\begin{aligned}
&\mathcal{T}[\![ListaAplicaciones]\!] \\
\equiv\ &\mathcal{T}[\![\ Aplicación_1\ ,\ \ldots\ ,\ Aplicación_n\ ]\!] \\
\equiv\ &\mathcal{T}[\![\ \underbrace{\epsilon}_{S'}\ ]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{Sd_1}\ ]\!]\ \mathcal{T}[\![\ Aplicación_1\ ]\!]\ \mathcal{T}[\![\ \underbrace{,}_{\widehat{Sc_1}}\ ]\!] \\
&\qquad\qquad \ldots\ \mathcal{T}[\![\ \underbrace{,}_{\widehat{Sc_{n-1}}}\ ]\!]\ \mathcal{T}[\![\ Aplicación_n\ ]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{Sd_2}\ ]\!] \\
\equiv\ &\mathcal{T}[\![\ Aplicación_1\ ]\!]\ ,\ \ldots\ ,\ \mathcal{T}[\![\ Aplicación_n\ ]\!]
\end{aligned}
\tag{4.29}
$$

– Traducción de una aplicación dentro de una lista de Aplicaciones:

$$
\begin{aligned}
&\mathcal{T}[\![\ Aplicación\ ]\!] \\
\equiv\ &\mathcal{T}[\![\ (\ Puerto\ )\ Identificador\ ]\!] \\
\equiv\ &\mathcal{T}[\![\ \underbrace{\epsilon}_{S'}\ ]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{Sd_1}\ ]\!]\ \mathcal{T}[\![\ \underbrace{(Prto)}_{X_1}\ ]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{\widehat{Sc_1}}\ ]\!]\ \mathcal{T}[\![\ \underbrace{Id.}_{X_2}\ ]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{Sd_2}\ ]\!] \\
\equiv\ &\mathcal{T}[\![\ Identificador\ ]\!]
\end{aligned}
\tag{4.30}
$$

- Definición de un objeto:

$$
\begin{aligned}
&\mathcal{T}[\![ParteLógicaProceso]\!] \\
\equiv\ &\mathcal{T}[\![RestriccionesObjeto\ \triangleright\ ListaMétodos]\!] \\
\equiv\ &\mathcal{T}[\![\ \underbrace{\epsilon}_{S'}\ ]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{Sd_1}\ ]\!]\ \mathcal{T}[\![RestriccionesObjeto]\!] \\
&\qquad\qquad \mathcal{T}[\![\ \underbrace{\triangleright}_{\widehat{Sc_1}}\ ]\!]\ \mathcal{T}[\![ListaMétodos]\!]\ \mathcal{T}[\![\ \underbrace{\epsilon}_{Sd_2}\ ]\!]
\end{aligned}
\tag{4.31}
$$

  - Traducción de las restricciones de objeto:

$$
\begin{aligned}
&\mathcal{T}[\![RestriccionesObjeto]\!] \\
\equiv\ &\mathcal{T}[\![\ (\ RestriccionesRecepción,\ RestriccionesDelegación)\ ]\!] \\
\equiv\ &\mathcal{T}[\![\ \underbrace{\epsilon}_{S'}\ ]\!]\ \mathcal{T}[\![\ \underbrace{(}_{Sd_1}\ ]\!]\ \mathcal{T}[\![\ RestriccionesRecepción\ ]\!]\ \mathcal{T}[\![\ \underbrace{,}_{\widehat{Sc_1}}\ ]\!] \\
&\qquad\qquad \mathcal{T}[\![\ RestriccionesDelegación\ ]\!]\ \mathcal{T}[\![\ \underbrace{)}_{Sd_2}\ ]\!]
\end{aligned}
\tag{4.32}
$$

    - Traducción de los símbolos de sincronización:

$$\mathcal{T}[\![\underbrace{,}_{\widehat{Sc_1}}]\!] = \begin{cases} , & \text{Si } \delta_{forward}{}^{a} \neq \emptyset, \\ \epsilon & \text{en otro caso.} \end{cases} \quad (4.33)$$

[a] Grupo de restricciones de delegación

$$\equiv (\ \mathcal{T}[\![RestriccionesRecepción]\!]\ \mathcal{T}[\![\ ,\ ]\!]\ \mathcal{T}[\![RestriccionesDelegación]\!]\ )$$

$$\equiv (\ \mathcal{T}[\![\ ListaRestricciones_1\ ]\!]\ \mathcal{T}[\![\ ,\ ]\!]\ \mathcal{T}[\![\ ListaRestricciones_2\ ]\!]\ )$$

La traducción de una serie de restricciones fue presentada en la ecuación 4.25 en la página 96.

- Traducción de una lista de elementos (métodos) de objeto:

$$\begin{aligned} &\mathcal{T}[\![ListaMétodos]\!] \equiv \mathcal{T}[\![\ [\ E_1\ \&\ \ldots\ \&\ E_n\ ]\ ]\!] \\ \equiv &\mathcal{T}[\![\underbrace{\epsilon}_{S'}]\!]\ \mathcal{T}[\![\underbrace{[}_{Sd_1}]\!]\ \mathcal{T}[\![E_1]\!]\ \mathcal{T}[\![\underbrace{\&}_{\widehat{Sc_1}}]\!]\ \ldots\ \mathcal{T}[\![\underbrace{\&}_{\widehat{Sc_{n-1}}}]\!]\ \mathcal{T}[\![E_n]\!]\ \mathcal{T}[\![\underbrace{]}_{Sd_2}]\!] \\ \equiv &[\ \mathcal{T}[\![E_1]\!]\ \&\ \ldots\ \&\ \mathcal{T}[\![E_n]\!]\ ] \end{aligned} \quad (4.34)$$

  - Traducción de un elemento (método) de objeto:

$$\begin{aligned} &\mathcal{T}[\![Método]\!] \\ \equiv &\mathcal{T}[\![Nombre\ :\ Parámetros\ Programa]\!] \\ \equiv &\mathcal{T}[\![\underbrace{Nombre\ :}_{S'}]\!]\ \mathcal{T}[\![\underbrace{\epsilon}_{Sd_1}]\!]\ \mathcal{T}[\![Parámetros]\!]\ \mathcal{T}[\![\underbrace{\epsilon}_{\widehat{Sc_1}}]\!] \\ &\qquad\qquad \mathcal{T}[\![Programa]\!]\ \mathcal{T}[\![\underbrace{\epsilon}_{Sd_2}]\!] \\ \equiv &\ Nombre\ :\ \mathcal{T}[\![Parámetros]\!]\ \mathcal{T}[\![Programa]\!] \end{aligned} \quad (4.35)$$

    - Traducción de una lista de Parámetros:

$$\begin{aligned} &\mathcal{T}[\![Parámetros]\!] \\ \equiv &\mathcal{T}[\![\ (\ ListaIdentificadores\ )\ ]\!] \\ \equiv &\mathcal{T}[\![\underbrace{\epsilon}_{S'}]\!]\ \mathcal{T}[\![\underbrace{(}_{Sd_1}]\!]\ \mathcal{T}[\![\ ListaIdentificadores\ ]\!]\ \mathcal{T}[\![\underbrace{)}_{Sd_1}]\!] \\ \equiv &(\ \mathcal{T}[\![\ ListaIdentificadores\ ]\!]\ ) \end{aligned} \quad (4.36)$$

El preorden $\left(\ell(\text{ GraPiCO }), \succ\right)$

Dibujando la relación $\succ$ resulta el grafo de la Figura 4.10, que constituye el Grafo Acíclico Dirigido (desde ahora GDA) para la gramática del cálculo GraPiCO.

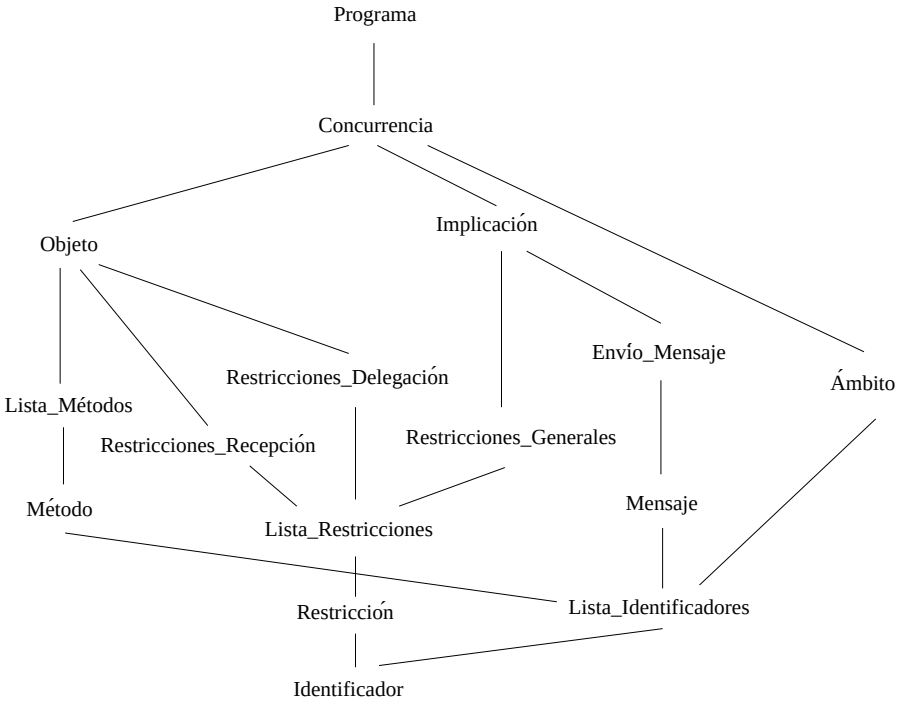


Figura 4.10: Grafo de la relación $\succ$ en ℓ(GraPiCO).

1. Se deduce por las reglas de traducción presentadas en la sección 4.3.1 y por el GDA presentado en la Figura 4.10 lo siguiente:

   - Para todos los constructores visuales simples:
     - La especificación gramatical del constructor X es de la forma:
       $\mathcal{E}[\![X]\!] = S'_X \; {Sd_1}_X \; PF_X \sim \; Name(X) \; : \; Y \; {Sd_2}_X$
     - El constructor X está compuesto por el constructor Y, Y menor que X según la relación $\succ$
       $\forall_{Y \in \; \mathcal{E}[\![X]\!]}. \; Y \in Ex(X) \rightarrow Y \in Ex(X)^* \rightarrow X \succ Y$
   - Para todo constructor visual compuesto L_X como el presentado en la Figura 4.11:

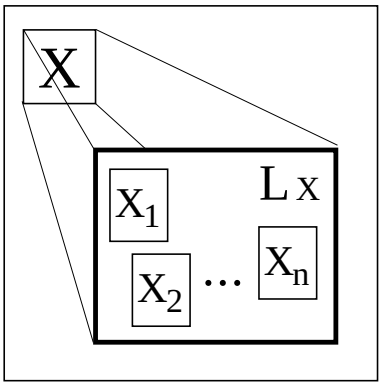


Figura 4.11: Constructor visual compuesto L_X y sus constructores X_1, X_2, $\cdots$ y X_n.

- La especificación gramatical de la lista de constructores LX es de la forma: $\mathcal{E}[\![LX]\!] = S'\ Sd_1\ X_1\ \widehat{Sc_1} \cdots \widehat{Sc_{n-1}}\ X_n\ Sd_2$
- La lista de constructores LX está compuesta de los constructores X_1, $\cdots$,X_n, todo X_i menor que LX según la relación $\succ$
  $\forall_{X_i \in\ \mathcal{E}[LX]}.\ X_i \in Ex(LX) \rightarrow X_i \in Ex(LX)^* \rightarrow LX \succ X_i$

2. A través del GDA se encuentra un árbol sintáctico (AS en adelante) para cada programa GraPiCO P_i. Lo anterior es expresado de forma simbólica en 4.37.

$$\forall_{P_i \in\ \ell(\text{ GraPiCO })} \exists_{AS_i} \tag{4.37}$$

3. Puesto que todo programa GraPiCO P_i es finito, su respectivo AS_i 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 4.12 la relación $\succ$ sobre ℓ(GraPiCO) es transitiva y por 4 no existen sucesiones infinitas estrictamente decrecientes, tenemos el preorden bien fundado (desde ahora pbf) $\left(\ell(\text{ GraPiCO }), \succ\right)$.

Teorema 1. (Principio de Inducción Noetheriana). *También denominado Principio de Inducción completa sobre Preórdenes Bien Fundados. Sea $(D, <)$ un pbf y $P(x)$ un predicado sobre los elementos x de D. Si es posible demostrar que, siempre que todos los predecesores estrictos b de cualquier elemento a de D cumplen el predicado P, y también lo cumple el propio a, entonces todos los elementos lo cumplen:*

$$\frac{\forall a \in D.\ (\forall b \in D.\ b \prec a \rightarrow P(b)) \rightarrow P(a)}{\forall a \in D.\ P(a)} \tag{4.38}$$

*La aplicación práctica del Principio de Inducción Noetheriana sigue los pasos habituales de cualquier demostración por inducción, Base **(B)**, Hipótesis **(H)** e Inducción **(I)**:*

(B) *Demostrar $P(m)$ para todo elemento minimal m de D.*

(H) *Dado un no minimal $a \in D$, suponer que $P(b)$ se cumple para todo elemento $b < a$.*

(I) *Bajo la hipótesis de inducción, demostrar que se cumple $P(a)$.*

y puede emplearse el principio de inducción noetheriana en la ecuación 4.8 a través de la fórmula 4.39.

$$\frac{\begin{array}{l}\forall_{X \in \ell(\text{ GraPiCO })}. \\ \Big(\forall_{Y \in \ell(\text{ GraPiCO })}. X \succ Y \to \mathcal{E}[\![Y]\!] \in \ell(\text{GraPiCO\_Tex.}) \Rightarrow \mathcal{T}\Big[\!\Big[\mathcal{E}[\![Y]\!]\Big]\!\Big] \in \ell(\text{ PiCO })\Big) \\ \Longrightarrow \mathcal{E}[\![X]\!] \in \ell(\text{ GraPiCO\_Textual }) \to \mathcal{T}\Big[\!\Big[\mathcal{E}[\![X]\!]\Big]\!\Big] \in \ell(\text{ PiCO })\end{array}}{\forall_{X \in \ell(\text{ GraPiCO })}. \mathcal{E}[\![X]\!] \in \ell(\text{ GraPiCO\_Textual }) \to \mathcal{T}\Big[\!\Big[\mathcal{E}[\![X]\!]\Big]\!\Big] \in \ell(\text{ PiCO })} \tag{4.39}$$

Empleando la ecuación 4.9 en 4.39 resulta en 4.40 una expresión más concreta para aplicar la inducción noetheriana.

$$\forall_{X \in \ell(\text{ GraPiCO\_}\mathcal{E}\text{ })}. \mathcal{T}\Big[\!\Big[\mathcal{E}[\![X]\!]\Big]\!\Big] \in \ell(\text{ PiCO }) \tag{4.40}$$

En la Figura 4.10 es visible que $Identificador$ es el único elemento minimal (de hecho es el mínimo) en el GDA de GraPiCO.

$$m \text{ es mínimo en } \ell \text{ (GraPiCO) } \stackrel{\text{def}}{=} \exists!_m. \neg\exists_x.\ m \succ x \tag{4.41}$$

Con todo lo anterior, es aplicable el principio de inducción noetheriana de la siguiente forma:

(***B***) Se demuestra para todos los elementos minimales (en este caso sólo *identificador*) que su traducción es un constructor PiCO.

$$\begin{aligned} & \mathcal{T}[\![Identificador\_GraPiCO]\!] \\ \equiv\ & \mathcal{T}[\![\ ParteFísica\ \sim\ Identificador\_PiCO\ ]\!] \\ \equiv\ & Identificador\_PiCO \end{aligned}$$

(***H***) Dado un X no minimal (para el caso debe ser diferente de $identificador$, el único minimal) se supone que la traducción de todos los constructores de los que esté compuesto X son constructores PiCO.

$$\forall_{X \in\ \ell(\text{GraPiCO\_}\mathcal{E}) \setminus Identificador}. \forall_{Y \prec X}. \mathcal{T}[\![Y]\!]\ \in\ \ell(\text{PiCO})$$

(***I***) Bajo la **Hipótesis**, se demuestra que la traducción de la especificación de todo constructor GraPiCO es un código PiCO.

$$\forall_{X \in \ell(\text{ GraPiCO\_}\mathcal{E}\text{ })}. \mathcal{T}\Big[\!\Big[\mathcal{E}[\![X]\!]\Big]\!\Big] \in \ell(\text{ PiCO })$$

Los constructores GraPiCO_textual son de la siguiente forma:

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

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

$\mathcal{T}[\![S']\!] \; \mathcal{T}[\![Sd_1]\!]$ PiCO′ $\mathcal{T}[\![Sd_2]\!]$ donde PiCO′ $\in \ell($ PiCO $)$

De la lista de reglas de traducción expuestas en la sección 4.3.1 de la página 93 están las siguientes funciones para transformar los símbolos de sincronización.

$$\mathcal{T}[\![S'_X]\!] = \begin{cases} S'_X & \text{Si } X = Mensaje,\ Método. \\ \mathcal{T}[\![clonación]\!] & \text{Si } X = Proceso. \\ \mathcal{T}[\![ClaseRestricciones]\!] & \text{Si } X = Restricciones. \\ \epsilon & \text{en otro caso.} \end{cases} \tag{4.42}$$

$$\mathcal{T}[\![Sd_{1_X}]\!] = \begin{cases} Sd_{1_X} & \text{Si } X = ParteLógicaPrograma,\ RestriccionesObjeto, \\ & \qquad Parámetros,\ Restricciones, \\ & \qquad Mensaje,\ ListaMétodos. \\ \textbf{\textit{local}} & \text{Si } X = Ámbito. \\ \epsilon & \text{en otro caso.} \end{cases} \tag{4.43}$$

$$\mathcal{T}[\![Sd_{2_X}]\!] = \begin{cases} Sd_{2_X} & \text{Si } X = ParteLógicaPrograma,\ RestriccionesObjeto, \\ & \qquad Parámetros,\ Restricciones, \\ & \qquad Mensaje,\ ListaMétodos. \\ . & \text{Si } X = Programa. \\ \epsilon & \text{en otro caso.} \end{cases} \tag{4.44}$$

En resumen, las reglas de traducción (presentando únicamente los resultados de la función $concatenación$ –rutina empleada para encadenar los resultados parciales dentro de $\mathcal{T}$, ver Figura 4.3 en la página 87–, mediante la utilización de las funciones presentadas en 4.42, 4.43 y 4.44, y la **Hipótesis**) es obtenida 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

- Implicación: AntecedentePiCO ***then*** ProgramaPiCO
  - Antecedente restricciones: ClaseRestriccionesPiCO ListaRestriccionesPICO
    - Lista de restricciones: **(** RestricciónPiCO_1 $\wedge \cdots \wedge$ RestricciónPiCO_n **)**
      - Restricción (tres direcciones): IdPiCO_1 OpPiCO IdPiCO_2 CompPiCO IdPiCO_3
        Restricción (dos direcciones): IdPiCO_1 CompPiCO IdPiCO_2
  - Antecedente envío mensaje: IdentificadorPiCO $\lhd$ mensajePiCO
    - NombreMensajePiCO **: [** ListaAplicacionesPiCO **]**
      - Lista de aplicaciones: AplicaciónPiCO_1 **,** $\cdots$ **,** AplicaciónPiCO_n
        - Aplicación: IdentificadorPiCO

- Objeto: RestriccionesObjetoPiCO $\rhd$ ListaMétodosPiCO
  - Restricciones de Objeto: **(** RestriccionesRecepciónPiCO **,** RestriccionesDelegaciónPiCO **)**
  - Lista de métodos: **[** MétodoPiCO_1 **&** $\cdots$ **&** MétodoPiCO_n **]**
    - Método: NombreMétodoPiCO **:** ParámetrosPiCO ProgramaPiCO
      - Parámetros: **(** ListaIdentificadoresPiCO **)**

Se puede ver que todos los resultados, en efecto, son constructores PiCO; terminando con esto la demostración.

4.3.2. Demostración de la *completitud* de la función de traducción $\mathcal{T}$

De la lista de Reglas de Traducción de la función $\mathcal{T}$ y las Fórmulas de especificación $\mathcal{E}$ se obtiene que las funciones $\mathcal{T}$ y $\mathcal{E}$ son biyectivas con los siguientes tipos.

- Reglas de Traducción $\mathcal{T}$: ℓ (GraPiCO) $\rightarrow \ell$ (GraPiCO_Textual)
- Fórmulas de especificación $\mathcal{E}$: ℓ (GraPiCO_Textual) $\rightarrow \ell$ (PiCO)

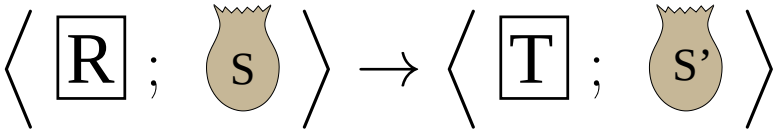
La composición de las funciones $\mathcal{T}$ y $\mathcal{E}$ es también una función biyectiva con el siguiente tipo:

- Composición de funciones $\mathcal{T} \circ \mathcal{E}$: ℓ (GraPiCO) $\rightarrow \ell$ (PiCO)

De ahora en adelante emplearemos las definiciones en 4.45 a continuación:

$$\begin{aligned} \boxed{\text{X}} &\stackrel{\text{def}}{=} X \in \ell(\text{ GraPiCO }) \\ X &\stackrel{\text{def}}{=} X \in \ell(\text{ PiCO }) \\ \left[\!\left[\boxed{\text{X}}\right]\!\right] &\stackrel{\text{def}}{=} \mathcal{T} \circ \mathcal{E} \left[\!\left[\boxed{\text{X}}\right]\!\right] \end{aligned} \tag{4.45}$$

Para poder establecer la *completitud* de la función de traducción se debe demostrar que si $[\![\boxed{R}]\!]$ = R reduce hacia algún T en PiCO entonces T es congruente con la codificación de algún $\boxed{T}$, tal que $\boxed{R}$ reduce hacia $\boxed{T}$, formalmente presentado en 4.46.



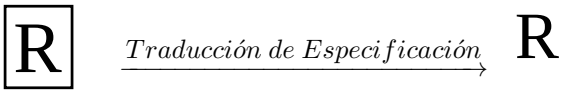
$$\langle R;\ S \rangle \rightarrow \langle T;\ S' \rangle \tag{4.46}$$

$$[\![\boxed{T}]\!] \equiv T$$

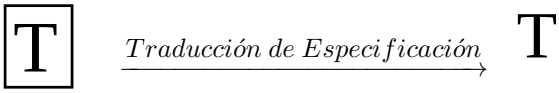
Dado que las relaciones de reducción de los cálculos PiCO y GraPiCO presentadas en la sección 3.7 configuran funciones inyectivas, se tiene que la reducción asigna imágenes distintas a constructores distintos del respectivo cálculo. No tomando en consideración la información del almacén de restricciones se tiene 4.47 para algunos $\boxed{R}$ y R y sus correspondientes $\boxed{T}$ y T.

$$\begin{array}{ccc} \boxed{R} & & R \\ \downarrow \textit{Reducción GraPiCO} & & \downarrow \textit{Reducción PiCO} \\ \boxed{T} & & T \end{array} \tag{4.47}$$

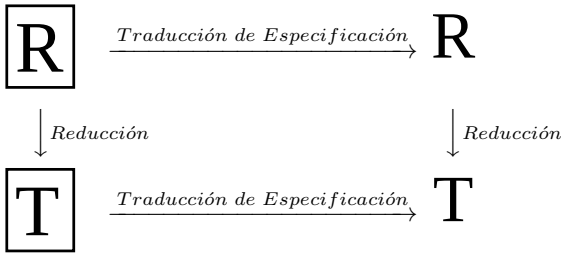
De la biyección de la composición de funciones $\mathcal{T} \circ \mathcal{E}$ se obtiene que para cada par de constructores GraPiCO diferentes existe una correspondencia con dos constructores PiCO diferentes, ver 4.48.



(4.48)

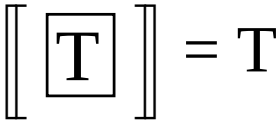


Con la unión de 4.47 y 4.48 resulta el diagrama conmutativo en 4.49.



(4.49)

Del diagrama conmutativo en 4.49 se infiere que todo constructor PiCO T corresponde a una codificación de un constructor GraPiCO $\boxed{\text{T}}$, resultando 4.50, lo que se requería. Empleando términos diferentes, la función de traducción hacia PiCO no produce computaciones que no corresponden a alguna computación GraPiCO; de esta forma termina la demostración.



(4.50)

Hasta el momento se ha determinado, por medio de las demostraciones de la *validez* y la *completitud*, que la función de traducción es *correcta*, pero aún no se ha establecido que los programas resultantes en PiCO despues de una traducción corresponden con el significado de los respectivos programas GraPiCO iniciales; por esto en la siguiente sección se presenta una demostración del cumplimiento de esta condición.

Capítulo 5

Prueba de consistencia de la semántica entre los programas GraPiCO y PiCO

Después de presentar las Reglas de Traducción GraPiCO_Textual-cálculo PiCO, se requiere la demostración de su corrección con base en la relación de reducción de los operadores visuales GraPiCO; lo anterior, con la intención de poder asegurar que los constructores visuales y la traducción de su especificación textual son consistentes con la semántica del cálculo PiCO; en otras palabras, demostrar la conservación de la semántica en el resultado que presentan las reglas de traducción de GraPiCO_Textual hacia cálculo PiCO. De esta forma, la prueba de corrección puede ser representada por el diagrama conmutativo mostrado en la Figura 5.1.

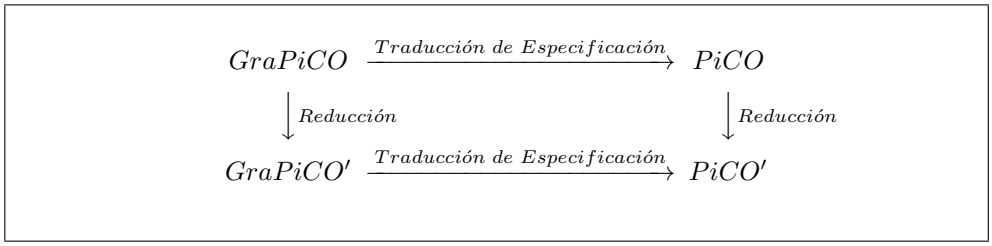


Figura 5.1: Diagrama conmutativo de la corrección de las reglas de traducción.

Siguiendo la Figura 5.1 el lector encontrará una prueba de corrección para cada constructor GraPiCO así:

- *Título de la prueba en letra itálica*
- Encabezados con números romanos los pasos siguientes:

  I. Lista de premisas (para la prueba) en forma visual y su explicación.

II. Desarrollo de la traducción del constructor GraPiCO hacia su constructor equivalente PiCO.

III. Realización de la conversión del operador visual GraPiCO' a su correspondiente constructor PiCO'.

IV. Construcción del diagrama conmutativo final con los resultados de II. y III.

A continuación se muestran las pruebas para cada constructor de GraPiCO:

Corrección del constructor nuevo ámbito para nombres.

I. Para la demostración de la conservación de la semántica de la traducción del constructor de nuevo ámbito para nombres, las premisas requeridas en el proceso de traducción son expuestas en la Figura 5.2 y se explican a continuación:

1. Se parte de la condición de una adecuada traducción de la especificación del conjunto de restricciones impuestas en el $Store$ visual. Lo anterior, siguiendo la ecuación de especificación mostrada en la Figura 3.34 de la página 64 y la regla de traducción presentada en la fórmula 4.25 en la página 96.
2. Luego de la inclusión de la restricción $v = v$ en el $Store$ visual, para la reducción del proceso GraPiCO P en el proceso P' se emplea la relación 6 de la Figura 3.61 en página 83.
3. Para la reducción de la creación de nuevo ámbito GraPiCO para el nombre x se utiliza la relación de reducción DEC-V en la Figura 3.55 de la página 76.

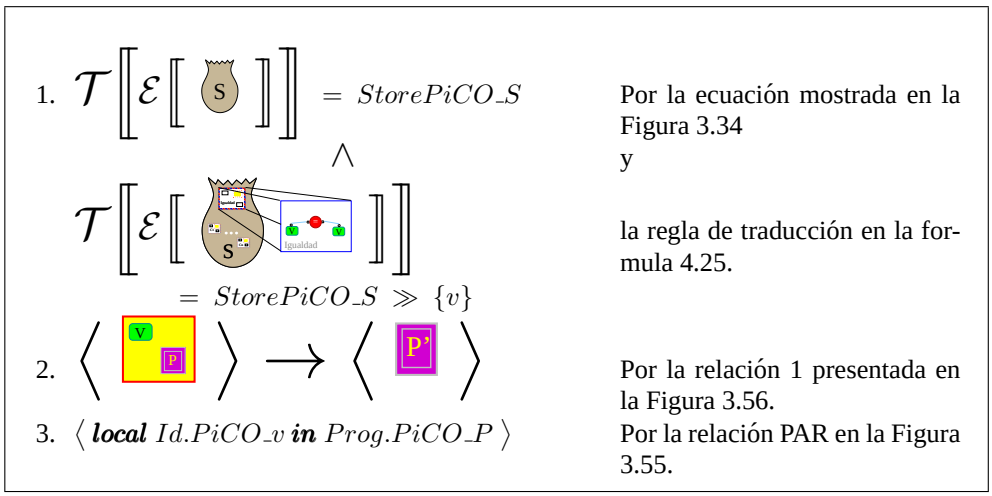


Figura 5.2: Premisas para el proceso de traducción.

II. Posterior a la presentación de las premisas, se procede con la traducción de la especificación del proceso de creación de nuevo ámbito para nombres o variables GraPiCO, así:

$\mathcal{T}\llbracket\ \underbrace{\mathcal{E}\llbracket\ \llbracket\ \text{V}\ \text{P}\ \rrbracket\ \rrbracket}_{1.}\ \rrbracket$

1. Especificación del operador de creación de ámbito con la fórmula en la Figura 3.13 de la pág. 55.

$= \mathcal{T}\llbracket\ [\ \underbrace{\mathcal{E}\llbracket\ G_v\ \rrbracket}_{2.}\ ;\ \underbrace{\mathcal{E}\llbracket\ G_n\ \rrbracket}_{2.}\ ;\ \underbrace{\mathcal{E}\llbracket\ P\ \rrbracket}_{3.}\ ]\ \rrbracket$

2. Especificación de un grupo de Identificadores GraPiCO con la fórmula en la Figura 3.12 de la pág. 55.
3. Especificación de programa a través de la fórmula en la Figura 3.7 de la página 53.

$= \mathcal{T}\llbracket\ [\ \underbrace{\mathcal{E}\llbracket\ v\ \rrbracket}_{4.}\ ;\ \emptyset\ ;\ \{\ ParteFísica_P \sim Etiqueta_P : ParteLógica_P\ \}\ ]\ \rrbracket$

4. Especificación de un Identificador GraPiCO mediante la fórmula en la Figura 3.15 de la pág. 56.

$= \underbrace{\mathcal{T}\llbracket\ [\ ParteFís_v \sim Etiqueta_v : ParteLógica_v\ ;\ \emptyset\ ;\ \{\ ParteFís_P \sim\ \cdots\ \}\ ]\ \rrbracket}_{5.}$

5. Traducción del operador de creación de ámbito con la regla 4.17 de la pág. 94.

$= \underbrace{\mathcal{T}\llbracket\ [\ \rrbracket}_{6.}\ \underbrace{\mathcal{T}\llbracket\ P.Fís_v \sim\ \cdots\ \rrbracket}_{7.}\ \underbrace{\mathcal{T}\llbracket\ ;\ \rrbracket}_{6.}\ \mathcal{T}\llbracket\ \emptyset\ \rrbracket\ \underbrace{\mathcal{T}\llbracket\ ;\ \rrbracket}_{6.}\ \underbrace{\mathcal{T}\llbracket\ \{\ P.Fís_P \sim\ \cdots\ \}\ \rrbracket}_{8.}\ \underbrace{\mathcal{T}\llbracket\ ]\ \rrbracket}_{6.}$

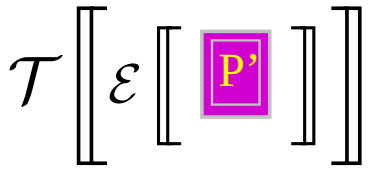
6. Traducción de los símbolos de sincronización de ámbito con la regla 4.18 de la página 95.
7. Traducción de un Identificador mediante la regla 4.20 de la página 95.
8. Traducción de programa por medio de la regla 4.13 de la página 93.

$= \textbf{\textit{local}}\ \mathcal{T}\llbracket\ ParteLógica\_v\ \rrbracket\ \textbf{\textit{in}}\ \mathcal{T}\llbracket\ ParteLógica\_P\ \rrbracket$

$= \textbf{\textit{local}}\ VariablePiCO\_v\ \textbf{\textit{in}}\ ProgramaPiCO\_P$

Figura 5.3: Resolución de la traducción de la especificación del operador visual de creación de nuevo ámbito.

III. De igual forma se procede a la traducción de la especificación del programa P'.



Especificación de programa a través de la fórmula en la Figura 3.7 de la página 53.

$= \mathcal{T}[\![\{ P.Fís_{P'} \,.\, Etiqueta_{P'} : P.Lóg_{P'} \}]\!]$

Traducción de un programa por medio de la regla 4.13 en la página 93.

$= ProgramaPiCO\_P'$

Figura 5.4: Resolución de la traducción de la especificación del constructor visual de programa.

IV. Luego de la presentación de las premisas en I y por los pasos II y III se puede construir el diagrama conmutativo de la Figura 5.5 muestra, en efecto, cómo los procesos de especificación $\mathcal{E}$ y traducción $\mathcal{T}$ conservan las relaciones de reducción de los operadores de creación de nuevo ámbito de GraPiCO y PiCO.

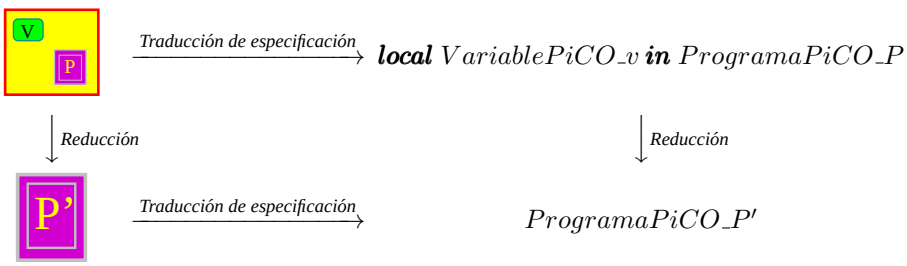


Figura 5.5: Diagrama conmutativo de la corrección de las reglas de traducción del constructor visual de creación de ámbito.

□

Corrección del constructor de concurrencia.

I. Para la demostración de la conservación de la semántica de la traducción del constructor de procesos concurrentes, las premisas requeridas en el proceso de traducción se presentan en la Figura 5.6 y son explicadas a continuación:

1. A partir de la condición de una adecuada traducción de la especificación del conjunto de restricciones impuestas en el $Store$ visual. Lo anterior, siguiendo la ecuación de especificación mostrada en la Figura 3.34 de la página 64 y la regla de traducción presentada en la fórmula 4.25 en la página 96.
2. Dado que el proceso GraPiCO P reduce hacia P', para la reducción de los procesos concurrentes GraPiCO, etiquetados como Q y P, se emplea la relación 1 presentada en la Figura 3.56 de la página 77.
3. Para la reducción de los Elementos concurrentes $ProcesoPiCO\_Q$ y $ProcesoPiCO\_P$ y, dado que $Proceso\_P$ reduce a $ProcesoPiCO\_P'$, se utiliza la relación de reducción PAR presentada en la Figura 3.55 de la página 76.

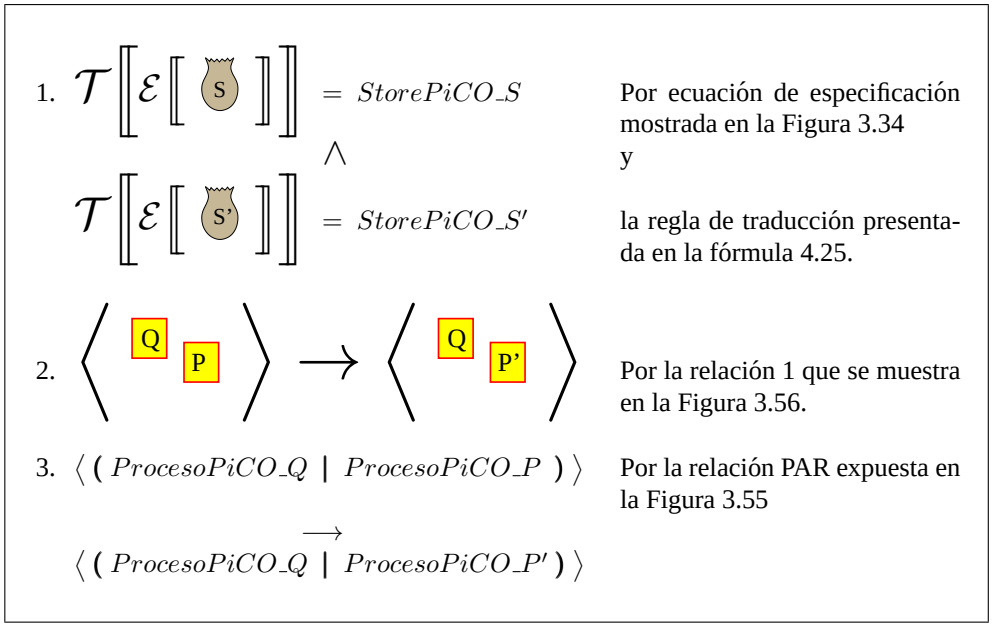


Figura 5.6: Premisas para el proceso de traducción.

II. Posterior a la presentación de las premisas se procede con la traducción de la especificación de los procesos concurrentes GraPiCO etiquetados como P y Q, así:

$$\mathcal{T}\Big[\!\Big[\underbrace{\mathcal{E}\Big[\!\Big[\boxed{Q} \quad \boxed{P} \Big]\!\Big]}_{1.} \Big]\!\Big]$$

1. Especificación del operador de concurrencia GraPiCO empleando la fórmula en la Figura 3.8 de la pág. 53.

$$= \mathcal{T}[\![\, (\, \underbrace{\mathcal{E}[\![\, Q \,]\!]}_{2.} \mid \underbrace{\mathcal{E}[\![\, P \,]\!]}_{2.} \,) \,]\!]$$

2. Especificación de procesos GraPiCO al utilizar la fórmula en la Figura 3.10 de la pág. 54.

$$= \underbrace{\mathcal{T}[\![\, (\, P.Fís_Q \,.\, Etiqueta_Q \;:\; P.Lóg_Q \;\mid\; P.Fís_P \,.\, Etiqueta_P \;:\; P.Lóg_P \;) \,]\!]}_{3.}$$

3. Traducción de operador de concurrencia GraPiCO_Textual con la regla 4.14 de la página 94.

$$= \big(\underbrace{\mathcal{T}[\, P.Fís_Q \,.\, Etiqueta_Q \;:\; P.Lóg_Q \,]}_{4.} \mid \underbrace{\mathcal{T}[\, P.Fís_P \,.\, Etiqueta_P \;:\; P.Lóg_P \,]}_{4.} \big)$$

4. Traducción de los procesos Q y P mediante la regla 4.15 de la página 94.

$$= (\, \mathcal{T}[\![\, ParteLógica_Q \,]\!] \mid \mathcal{T}[\![\, ParteLógica_P]\!] \,)$$

Traducción de procesos GraPiCO_Textual con la regla 4.15 de la pág. 94.

$$= (\, ProcesoPiCO\_Q \mid ProcesoPiCO\_P \,)$$

Figura 5.7: Resolución de la traducción de la especificación del constructor visual de concurrencia 1.

III. Al igual, se aplica el mismo procedimiento aplicado a los procesos concurrentes P y Q en la Figura 5.7, sobre los procesos concurrentes P' y Q. Un resumen de este tratamiento se presenta en la Figura 5.8.

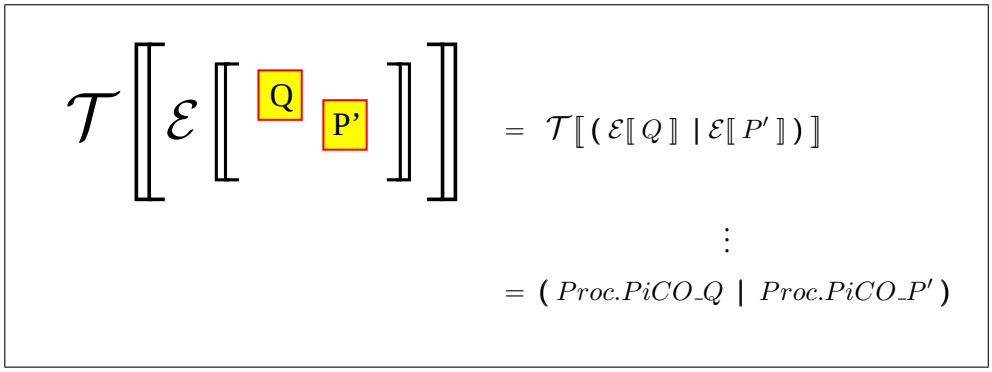


Figura 5.8: Resolución de la traducción de la especificación del constructor visual de concurrencia 2.

IV. Luego de la presentación de las premisas en I y por los pasos II y III, es posible construir el diagrama conmutativo de la Figura 5.9 que muestra, en efecto, cómo los procesos de especificación $\mathcal{E}$ y traducción $\mathcal{T}$ conservan las relaciones de reducción de los procesos concurrentes.

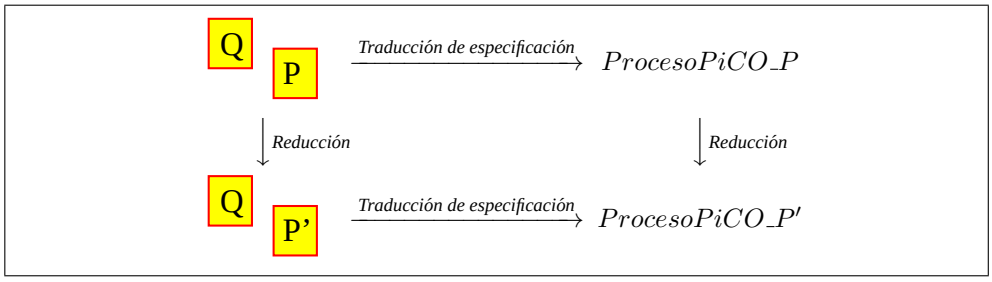


Figura 5.9: Diagrama conmutativo de la corrección de las reglas de traducción del constructor de concurrencia.

□

Corrección del constructor de imposición de restricciones.

I. Para la demostración de la conservación de la semántica de la traducción del constructor de imposición de restricciones, las premisas requeridas se presentan en la Figura 5.10 y son explicadas a continuación:

1. Se parte de la condición de una adecuada traducción de la especificación de los siguientes casos:
   a) Conjunto de restricciones impuestas en el $Store$ visual.
   b) Conjunto de restricciones impuestas en el $Store$ visual junto con la adición de un grupo de restricciones G_c.

Lo anterior, siguiendo la ecuación de especificación mostrada en la Figura 3.34 de la página 64 y la regla de traducción presentada en la fórmula 4.25 en la página 96.

2. Para la reducción del proceso implicación GraPiCO I con el antecedente T (imposición del conjunto de restricciones G_c) y el consecuente P (programa) se emplea la relación 5 presentada en la Figura 3.60 de la página 82.
3. Para la reducción del proceso implicación $ImplicaciónPiCO\_I$ con el antecedente ***tell*** ϕ y el consecuente $ProgramaPiCO\_P$ se utiliza la relación TELL del sistema de transición presentado en la Figura 3.55 de la página 76.

1. a) $\mathcal{T}[\![\mathcal{E}[\![$  $]\!]]\!] = StorePiCO\_S$

Por la ecuación de especificación en la Figura 3.34 y la regla de traducción

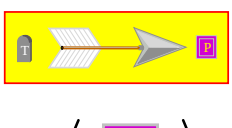
b) $\mathcal{T}[\![\mathcal{E}[\![$  $]\!]]\!]$

presentada en la fórmula 4.25.

$= StorePiCO\_S \wedge \underbrace{\mathcal{T}[\![\mathcal{E}[\![C_1]\!] \wedge \cdots \wedge \mathcal{E}[\![C_n]\!]]\!]}_{\phi}$

Por definición de ϕ.

$= StorePiCO\_S \wedge \phi$

2. $\langle$  $\rangle$

$\longrightarrow \langle$  $\rangle$

Por la relación 5 en la Figura 3.60.

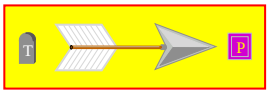
3. $\langle$ ***tell*** ϕ ***then*** $ProgramaPiCO\_P \rangle$

$\longrightarrow \langle\ ProgramaPiCO\_P \rangle$

Por la relación TELL en la Figura 3.55.

Figura 5.10: Premisas para el proceso de traducción.

II. Posterior a la presentación de las premisas se procede con la traducción de la especificación del proceso de imposición de las restricciones identificadas como ϕ, así:

$$\mathcal{T}\Big[\!\Big[\underbrace{\mathcal{E}\Big[\!\Big[\ \text{  }\ \Big]\!\Big]}_{1.}\Big]\!\Big]$$

1. Especificación de implicación con la fórmula en la fig. 3.27 de la pág. 60.

$$= \mathcal{T}\big[\!\big[\,\langle\ \underbrace{\mathcal{E}\big[\!\big[\,T\,\big]\!\big]}_{2.}\ ;\ \underbrace{\mathcal{E}\big[\!\big[\,P\,\big]\!\big]}_{3.}\ \rangle\,\big]\!\big]$$

2. Especificación de antecedente, con la fórmula en la Figura 3.26 de la pág. 60.
3. Especificación de consecuente (Prog.) por la regla en la fig. 3.7 de la pág. 53.

$$= \mathcal{T}\big[\!\big[\,\langle\,!\ \underbrace{\mathcal{E}\big[\!\big[\,Ex(T)\,\big]\!\big]}_{4.}\ ;\{\ ParteFísica_P\ .\ Etiqueta_P :\ ParteLógica_P\ \}\,\rangle\,\big]\!\big]$$

4. Especificación de la exp. de la imposición de restricciones con la fórmula en la fig. 3.34 de la pág. 64.

$$= \underbrace{\mathcal{T}\big[\!\big[\,\langle\,!\,\mathcal{E}\big[\!\big[\,C_1\,\big]\!\big]\ \wedge\cdots\wedge\ \mathcal{E}\big[\!\big[\,C_n\,\big]\!\big]\,;\{\ P.Física_P\ .\ Etiqueta_P :\ ParteLógica_P\ \}\,\rangle\,\big]\!\big]}_{5.}$$

5. Traducción de una implicación mediante la regla 4.21 de la página 95.

$$= \underbrace{\mathcal{T}\big[\!\big[\,!\,\mathcal{E}\big[\!\big[\,C_1\,\big]\!\big]\ \wedge\cdots\wedge\ \mathcal{E}\big[\!\big[\,C_n\,\big]\!\big]\,\big]\!\big]}_{6.}\ \textbf{\textit{then}}\ \underbrace{\mathcal{T}\big[\!\big[\,\{\ P.Fís_P\ .\ Etiqueta_P :\ P.Lóg_P\ \}\,\big]\!\big]}_{7.}$$

6. Traducción de un antecedente de imposición de restricciones según la regla 4.23 de la página 96.
7. Traducción de un programa a través de la regla 4.13 en la página 93.

$$= \underbrace{\mathcal{T}\big[\!\big[\,!\,\big]\!\big]}_{8.}\ \underbrace{\mathcal{T}\big[\!\big[\,\mathcal{E}\big[\!\big[\,C_1\,\big]\!\big]\ \wedge\cdots\wedge\ \mathcal{E}\big[\!\big[\,C_n\,\big]\!\big]\,\big]\!\big]}_{9.}\ \textbf{\textit{then}}\ ProgramaPiCO\_P$$

8. Traducción de la clase de restricciones a través de la regla 4.24 de la pág. 96.
9. Utilizando la definición en la premisa 1. b) de la Figura 5.10 en la página 114.

$$= \textbf{\textit{Tell}}\ \phi\ \textbf{\textit{then}}\ ProgramaPiCO\_P$$

Figura 5.11: Resolución de la traducción de la especificación de imposición de restricciones.

III. De igual forma, se procede a la traducción de la especificación del consecuente (programa) P.

$$\mathcal{T}[\![\mathcal{E}[\![\ \boxed{\text{P}}\ ]\!]]\!]$$

Especificación de consecuente (programa) a través de la fórmula en la Figura 3.7 de la página 53.

$$= \mathcal{T}[\![\ \{\ P.Física_P\ .\ Etiqueta_P\ :\ P.Lógica_P\ \}\ ]\!]$$

Traducción de un programa a través de la regla 4.13 en la página 93.

$$= ProgramaPiCO\_P$$

Figura 5.12: Resolución de la traducción de la especificación del constructor visual de programa.

IV. Luego de la presentación de las premisas en I y por los pasos II y III se construye el diagrama conmutativo de la Figura 5.13 que muestra, en efecto, cómo los procesos de especificación $\mathcal{E}$ y traducción $\mathcal{T}$ conservan las relaciones de reducción del constructor de imposición de restricciones.

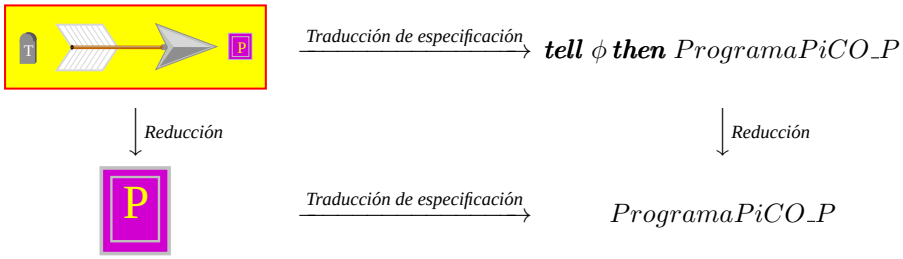


Figura 5.13: Diagrama conmutativo de la corrección de las reglas de traducción del constructor visual de imposición de restricciones.

□

Corrección del constructor de consulta de restricciones.

I. Para la demostración de la conservación de la semántica de la traducción del constructor de consulta de restricciones las premisas requeridas se presentan en la Figura 5.14 y se explican a continuación:

1. Se parte de la condición de una adecuada traducción de la especificación del conjunto de restricciones impuestas en el $Store$ visual.

   Lo anterior, sigue a la ecuación de especificación mostrada en la Figura 3.34 de la página 64 y la regla de traducción presentada en la fórmula 4.25 en la página 96.

2. Para la reducción del proceso implicación GraPiCO I con el antecedente T (imposición del conjunto de restricciones G_c) y el consecuente P (programa) se emplea la relación 5 presentada en la Figura 3.60 de la página 82.

3. Para la reducción del proceso implicación $ImplicaciónPiCO\_I$ con el antecedente ***ask*** ϕ y el consecuente $ProgramaPiCO\_P$ se utiliza la relación ASK del sistema de transición presentado en la Figura 3.55 de la página 76.

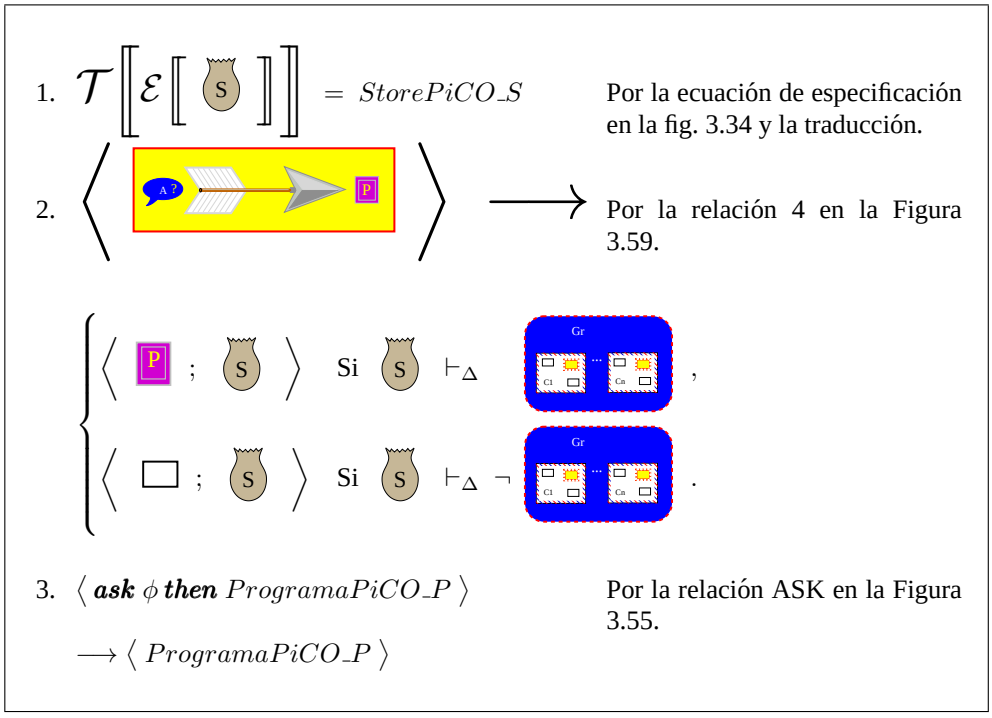
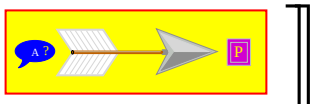


Figura 5.14: Premisas para el proceso de traducción.

II. Posterior a la presentación de las premisas, se procede con la traducción de la especificación del proceso de consulta de las restricciones identificadas como ϕ así:

$$\mathcal{T}[\![\ \underbrace{\mathcal{E}[\![\ \text{  }\ ]\!]}_{1.}\ ]\!]$$

1. Especificación del operador de implicación GraPiCO con la fórmula en la Figura 3.27 de la página 60.

$$= \mathcal{T}[\![\ \langle\ \underbrace{\mathcal{E}[\![\ T\ ]\!]}_{2.}\ ;\ \underbrace{\mathcal{E}[\![\ P\ ]\!]}_{3.}\ \rangle\ ]\!]$$

2. Especificación de antecedente con la fórmula en la Figura 3.26 de la pág. 60.
3. Esp. de consecuente (Prog) a través de la regla en la fig. 3.7 de la pág. 53.

$$= \mathcal{T}[\![\ \langle\ ?\ \underbrace{\mathcal{E}[\![\ Ex(T)\ ]\!]}_{4.}\ ;\ \{\ ParteFísica_P\ .\ Etiqueta_P :\ ParteLógica_P\ \}\ \rangle\ ]\!]$$

4. Esp. de la exp. de la Cons. de restricciones con la fórmula 3.34 de la pág. 64.

$$= \underbrace{\mathcal{T}[\![\ \langle\ ?\ \mathcal{E}[\![C_1]\!]\ \wedge \cdots \wedge\ \mathcal{E}[\![C_n]\!]\ ;\ \{\ P.Física_P\ .\ Etiqueta_P :\ P.Lógica_P\ \}\ \rangle\ ]\!]}_{5.}$$

5. Traducción de una implicación mediante la regla 4.21 de la página 95.

$$= \underbrace{\mathcal{T}[\![\ ?\ \mathcal{E}[\![C_1]\!]\ \wedge \cdots \wedge\ \mathcal{E}[\![C_n]\!]\ ]\!]}_{6.}\ \textbf{\textit{then}}\ \underbrace{\mathcal{T}[\![\ \{\ P.Fís_P\ .\ Etiqueta_P :\ P.Lóg_P\ \}\ ]\!]}_{7.}$$

6. Trad. de un antecedente de Cons. de restricciones con 4.23 de la pág. 96.
7. Traducción de un programa a través de la regla 4.13 en la página 93.

$$= \underbrace{\mathcal{T}[\![\ ?\ ]\!]}_{8.}\ \underbrace{\mathcal{T}[\![\ \mathcal{E}[\![\ C_1\ ]\!]\ \wedge \cdots \wedge\ \mathcal{E}[\![\ C_n\ ]\!]\ ]\!]}_{9.}\ \textbf{\textit{then}}\ ProgramaPiCO\_P$$

8. Traducción de la clase de restricciones a través de la regla 4.24 de la pág. 96.
9. Utilizando $\phi \stackrel{\text{def}}{=} \mathcal{T}[\![\ \mathcal{E}[\![\ C_1\ ]\!] \wedge \cdots \wedge \mathcal{E}[\![\ C_n\ ]\!]\ ]\!]$.

$$= \textbf{\textit{Ask}}\ \phi\ \textbf{\textit{then}}\ ProgramaPiCO\_P$$

Figura 5.15: Resolución de la traducción de la especificación del constructor visual de consulta de restricciones 1.

III. De igual forma, se procede a la traducción de la especificación del consecuente (programa) P.

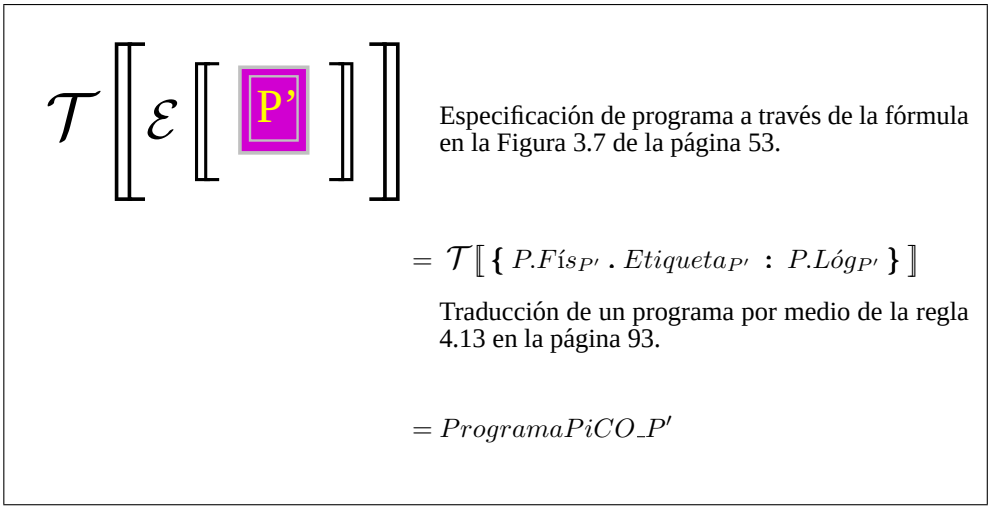


Figura 5.16: Resolución de la traducción de la especificación del constructor visual de programa.

IV. Luego de la presentación de las premisas en I y por los pasos II y III se construye el diagrama conmutativo de la Figura 5.17 que muestra, en efecto, como los procesos de especfificación $\mathcal{E}$ y traducción $\mathcal{T}$ conservan las relaciones de reducción del constructor de consulta de restricciones.

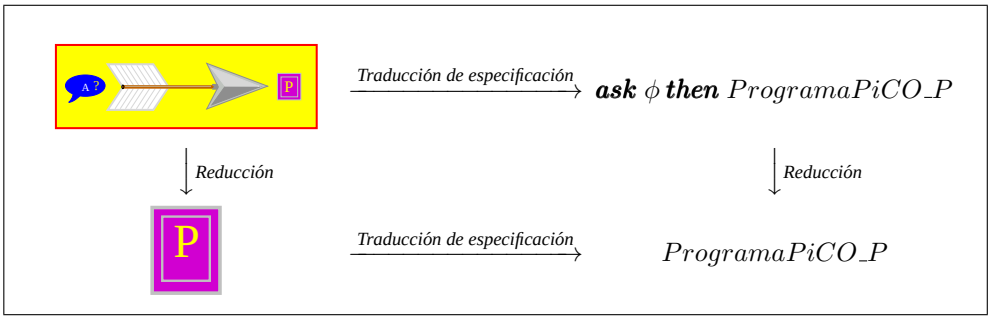


Figura 5.17: Diagrama conmutativo de la corrección de las reglas de traducción del constructor visual de consulta de restricciones.

□

Corrección del constructor de comunicación.

I. Para la demostración de la conservación de la semántica de la traducción del constructor de comunicación entre procesos, las premisas requeridas se presentan en la Figura 5.18 y se explican a continuación:

1. Se parte de la condición de una adecuada traducción de la especificación del conjunto de restricciones impuestas en el $Store$ visual. Lo anterior, siguiendo la ecuación de especificación mostrada en la Figura 3.34 de la página 64 y la regla de traducción presentada en la fórmula 4.25 en la página 96.
2. Para la reducción de una configuración con comunicación entre procesos GraPiCO etiquetados como E (envío de mensaje) y O (objeto) y, dado que las restricciones de recepción C del objeto O pueden ser derivadas del $Store$ S, se emplea la relación de transición 2 de la Figura 3.57 en la página 78.
3. Para la reducción de la comunicación entre I' y el objeto O y, dada la premisa 2., se utiliza la relación COMM del sistema de transición presentado en la Figura 3.55 de la página 76.

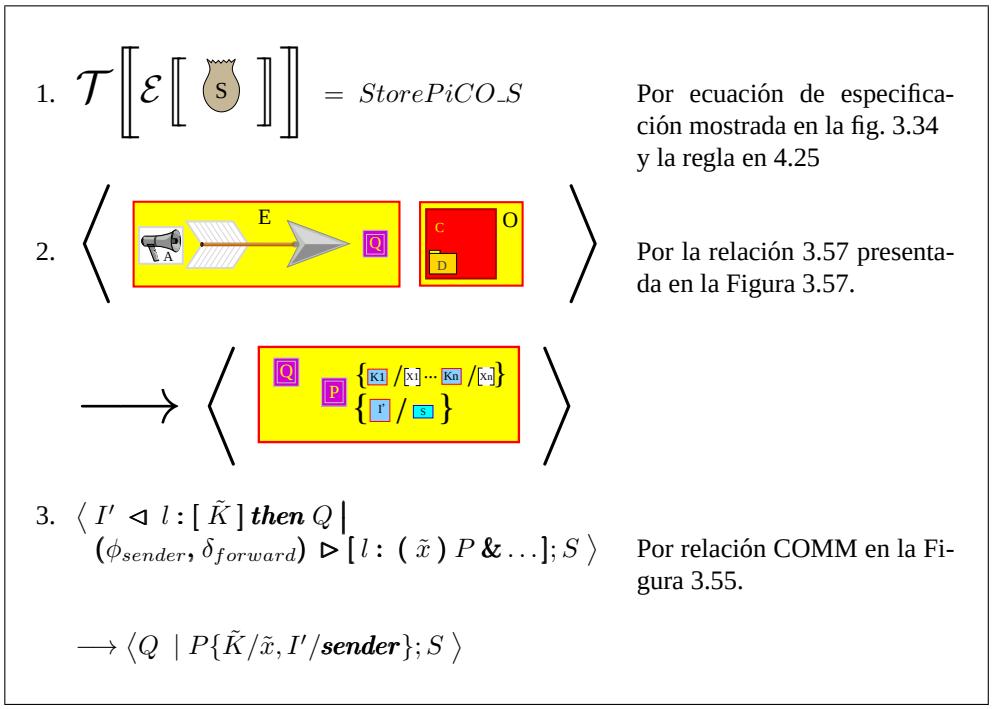
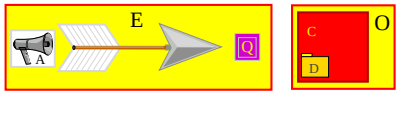


Figura 5.18: Premisas para el proceso de traducción.

II. Posterior a la presentación de las premisas se procede con la traducción de la especificación de la comunicación entre los procesos: envío de mensaje etiquetado como E y un objeto etiquetado como O, así:

$$\mathcal{T}\Big[\!\Big[\ \mathcal{E}\Big[\!\Big[\ \underbrace{\text{ 
}_{1.}\ \Big]\!\Big]\ \Big]\!\Big]$$

1. Especificación de la concurrencia empleando la fórmula 3.8 de la pág. 53.

$$= \mathcal{T}\Big[\!\Big[\ (\ \underbrace{\mathcal{E}[\![\ E\ ]\!]}_{2.}\ |\ \underbrace{\mathcal{E}[\![\ O\ ]\!]}_{3.}\ )\ \Big]\!\Big]$$

2. Especificación de implicación con la fórmula en la fig. 3.27 de la pág. 60.
3. Especificación objeto empleando la fórmula en la fig. 3.16 de la pág. 56.

$$= \mathcal{T}\Big[\!\Big[\ (\langle\ \underbrace{\mathcal{E}[\![\ A\ ]\!]}_{4.}\ ;\ \underbrace{\mathcal{E}[\![\ Q\ ]\!]}_{5.}\ \rangle\ |\ \underbrace{\mathcal{E}[\![\ Ex(C)\ ]\!]}_{6.} \rhd \underbrace{\mathcal{E}[\![\ Ex(D)\ ]\!]}_{7.}\ )\ \Big]\!\Big]$$

4. Especificación de antecedente, con la fórmula en la fig. 3.26 de la pág. 60, y especificación de envío de mensajes con la regla 3.32 de la pág. 63.
5. Esp. de consecuente (Prog) a través de la fórmula en la fig. 3.7 de la pág. 53.
6. Esp. de las restricciones de objeto por la fórmula de la fig. 3.25 en la pág. 59.
7. Especificación de Def. de objeto por la fórmula de la fig. 3.17 en la pág. 57.

$$= \mathcal{T}\Big[\!\Big[\ (\langle\ \underbrace{\mathcal{E}[\![\ I'\ ]\!]}_{8.} \lhd Name(M)\ :\ \underbrace{\mathcal{E}[\![\ G_r\ ]\!]}_{9.}\ ;\{\ P.Fís_Q\ .\ Etiqueta_Q :\ P.Lóg_Q\ \}\ \rangle$$
$$|\ (\ \underbrace{\mathcal{E}[\![\ Ex(R)\ ]\!]}_{10.}\ ,\ \underbrace{\mathcal{E}[\![\ Ex(D)\ ]\!]}_{10.}\ ) \rhd [\ \underbrace{\mathcal{E}[\![\ M\ ]\!]}_{11.}\ ])\ \Big]\!\Big]$$

8. Especificación de Identificadores con la fórmula de la fig. 3.15 de la pág. 56.
9. Esp. de un grupo de Relaciones con la regla de la fig. 3.29 de la pág. 62.
10. Esp. de un grupo de restricciones con la fórmula de la fig. 3.34 en la pág. 64.
11. Esp. de una Lista de Elementos por la regla en la fig. 3.19 de la pág. 57.

Figura 5.19: Resolución de la traducción de la especificación del constructor comunicación.

$$= \mathcal{T}[\![(\langle\, ParteFísica_{I'} \,.\, \underbrace{Name(I')}_{12.} \lhd Etiqueta_M \,:\, [\, \underbrace{\mathcal{E}[\![\, R_1 \,]\!]}_{13.}, \ldots, \underbrace{\mathcal{E}[\![\, R_n \,]\!]}_{13.} \,]$$
$$;\{\, ParteFísica_Q \,.\, Etiqueta_Q : ParteLógica_Q \,\}\,\rangle$$

$$| \,((\,\mathcal{E}[\![\, C_{\phi_1} \,]\!] \wedge \ldots \wedge \mathcal{E}[\![\, C_{\phi_i} \,]\!]\,), (\,\mathcal{E}[\![\, C_{\delta_1} \,]\!] \wedge \ldots \wedge \mathcal{E}[\![\, C_{\delta_j} \,]\!]\,))$$

$$\rhd\, [\, \underbrace{\mathcal{E}[\![\, E_1 \,]\!]}_{14.} \,\&\, \ldots \,\&\, \underbrace{\mathcal{E}[\![\, E_m \,]\!]}_{14.} \,])\,]\!]$$

12. Asignación de etiqueta impuesta por el programador.
13. Especificación de una relación de aplicación por medio de la fórmula en la Figura 3.30 de la página 62.
14. Especificación de un método con la fórmula en la Figura 3.14 de la página 56.

$$= \mathcal{T}[\![(\langle\, ParteFísica_{I'} \,.\, Etiqueta_{I'} \lhd Etiqueta_M$$
$$: [\,(\, \underbrace{\mathcal{E}[\![\, A_1 \,]\!]}_{15.} \,)\, \underbrace{\mathcal{E}[\![\, K_1 \,]\!]}_{16.}, \ldots, (\, \underbrace{\mathcal{E}[\![\, A_1 \,]\!]}_{15.} \,)\, \underbrace{\mathcal{E}[\![\, K_n \,]\!]}_{16.} \,]$$
$$;\{ ParteFísica_Q \,.\, Etiqueta_Q : ParteLógica_Q \,\}\rangle$$

$$| \,((\, \underbrace{\mathcal{E}[\![\, C_{\phi_1} \,]\!]}_{17.} \wedge \ldots \wedge \underbrace{\mathcal{E}[\![\, C_{\phi_i} \,]\!]}_{17.} \,), (\, \underbrace{\mathcal{E}[\![\, C_{\delta_1} \,]\!]}_{17.} \wedge \ldots \wedge \underbrace{\mathcal{E}[\![\, C_{\delta_j} \,]\!]}_{17.} \,))$$
$$\rhd\, [\, Etiqueta_{M'_1} : \underbrace{\mathcal{E}[\![\, M'_1 \,]\!]}_{18.} \,\&\, \ldots \,\&\, Etiqueta_{M'_m} : \underbrace{\mathcal{E}[\![\, M'_m \,]\!]}_{18.} \,])\,]\!]$$

15. Especificación de un puerto con la fórmula de la Figura 3.31 en la página 63.
16. Especificación de Identificadores con la fórmula de la Figura 3.15 de la página 56.
17. Especificación de las restricciones mediante las fórmulas de la Figura 3.35 en la página 65.
18. Especificación del cuerpo de un método según la fórmula de la Figura 3.23 en la página 59.

Figura 5.20: Resolución de la traducción de la especificación del constructor comunicación. (continuación a)

$$
\begin{aligned}
= \mathcal{T}[\![\, (\langle \, & ParteFísica_{I'} \,.\, Etiqueta_{I'} \lhd Etiqueta_M \;: \\
& [\, (\, ParteFísica_{A_1} \,.\, \underbrace{Name(A_1)}_{19.} \,)\, ParteFísica_{K_1} \,.\, \underbrace{Name(K_1)}_{19.} \,, \ldots \,] \\
& ; \{ ParteFísica_Q \,.\, Etiqueta_Q : ParteLógica_Q \} \rangle \\
& \\
| \; & (\, (\, RestricciónGraPiCO\_C_{\phi_1} \wedge \ldots \,) \,, (\, Restricción\_C_{\delta_1} \wedge \ldots \,) \,) \\
& \rhd [\, Etiqueta_{M'_1} : \underbrace{\mathcal{E}[\![\, G_{a\ M'_1} \,]\!]}_{20.} \; \underbrace{\mathcal{E}[\![\, P_{M'_1} \,]\!]}_{21.} \;\&\; \ldots \,] \,) \,]\!]
\end{aligned}
$$

19. Asignación de etiqueta impuesta por el programador.
20. Esp. del grupo de argumentos por la fórmula de la fig. 3.22 en la pág. 58.
21. Esp. del cuerpo de método (Prog.) con la regla en la fig. 3.7 de la pág. 53.

$$
\begin{aligned}
= \mathcal{T}[\![\, (\langle \, & ParteFísica_{I'} \,.\, Etiqueta_{I'} \lhd Etiqueta_M \;: \\
& [\, (\, ParteFísica_{A_1} \,.\, Etiqueta_{A_1} \,)\, ParteFísica_{K_1} \,.\, Etiqueta_{K_1} \,, \ldots \,] \\
& ; \{ ParteFísica_Q \,.\, Etiqueta_Q : ParteLógica_Q \} \rangle \\
& \\
| \; & (\, (\, RestricciónGraPiCO\_C_{\phi_1} \wedge \ldots \,) \,, (\, Restricción\_C_{\delta_1} \wedge \ldots \,) \,) \\
& \rhd [\, Etiqueta_{M'_1} : (\, \underbrace{\mathcal{E}[\![\, X_1 \,]\!]}_{22.} \,, \cdots \,, \underbrace{\mathcal{E}[\![\, X_n \,]\!]}_{22.} \,) \\
& \{ \, ParteFísica_P \,.\, Etiqueta_P \;:\; ParteLógica_P \, \} \;\&\; \ldots \,] \,) \,]\!]
\end{aligned}
$$

22. Esp. de argumentos con la fórmula en la Figura 3.24 de la página 59.

$$
\left.
\begin{aligned}
= \mathcal{T}[\![\, (\langle \, & ParteFísica_{I'} \,.\, Etiqueta_{I'} \lhd Etiqueta_M \;: \\
& [\, (\, P.Física_{A_1} \,.\, Etiqueta_{A_1} \,)\, P.Física_{K_1} \,.\, Etiqueta_{K_1} \,, \ldots \,] \\
& ; \{ ParteFísica_Q \,.\, Etiqueta_Q : ParteLógica_Q \} \rangle \\
& \\
| \; & (\, (\, RestricciónGraPiCO\_C_{\phi_1} \wedge \ldots \,) \,, (\, Restricción\_C_{\delta_1} \wedge \ldots \,) \,) \\
& \rhd [\, Etiqueta_{M'_1} : (\, ParteFísica_{X_1} \,.\, Etiqueta_{X_1} \,, \cdots \,) \\
& \{ \, ParteFísica_P \,.\, Etiqueta_P \;:\; ParteLógica_P \, \} \;\&\; \ldots \,] \,) \,]\!]
\end{aligned}
\right\} 23.
$$

23. Traducción de procesos concurrentes empleando la regla 4.14 de la página 94.

Figura 5.21: Resolución de la traducción de la especificación del constructor comunicación. (continuación b)

$$
\begin{array}{l}
= (\\
\quad \left.\begin{array}{l}
\mathcal{T}[\![\langle\, ParteFísica_{I'} \,.\, Etiqueta_{I'} \lhd Etiqueta_M \,: \\
\quad [\,(ParteFísica_{A_1} \,.\, Etiqueta_{A_1})\; ParteFísica_{K_1} \,.\, Etiqueta_{K_1}\,,\, \ldots\,] \\
\quad ;\{ ParteFísica_Q \,.\, Etiqueta_Q : ParteLógica_Q \}\,\rangle\,]\!]
\end{array}\right\} 24. \\
\quad | \\
\quad \left.\begin{array}{l}
\mathcal{T}[\![(\,(RestricciónGraPiCO\_C_{\phi_1} \wedge \ldots)\,,\,(\,Restricción\_C_{\delta_1} \wedge \ldots\,)\,) \\
\quad \rhd [\; Etiqueta_{M'_1} : (\; ParteFísica_{X_1} \,.\, Etiqueta_{X_1}\,,\, \cdots\,) \\
\quad \{\; ParteFísica_P \,.\, Etiqueta_P : ParteLógica_P \,\}\,\&\, \ldots\,]\,]\!]
\end{array}\right\} 25. \\
)
\end{array}
$$

24. Traducción del proceso implicación mediante la regla 4.21 de la página 95.
25. Traducción del proceso objeto a través de la regla 4.31 en la página 98.

$$
\begin{array}{l}
= (\\
\quad \left.\begin{array}{l}
\mathcal{T}[\![\; ParteFísica_{I'} \,.\, Etiqueta_{I'} \lhd Etiqueta_M \,: \\
\quad [(ParteFísica_{A_1} \,.\, Etiqueta_{A_1}) ParteFísica_{K_1} \,.\, Etiqueta_{K_1}\,,\, \ldots\,]\,]\!]
\end{array}\right\} 26. \\
\quad \textbf{then} \\
\quad \underbrace{\mathcal{T}[\![\,\{ ParteFísica_Q \,.\, Etiqueta_Q : ParteLógica_Q \}\,]\!]}_{27.} \\
\quad | \\
\quad (\, \underbrace{\mathcal{T}[\![\,(\; Restricción\_C_{\phi_1} \wedge \ldots\,)\,]\!]}_{28.}\,,\, \underbrace{\mathcal{T}[\![\,(\; Restricción\_C_{\delta_1} \wedge \ldots\,)\,]\!]}_{28.}\,) \\
\qquad \rhd [\\
\qquad\quad \left.\begin{array}{l}
\mathcal{T}[\![\; Etiqueta_{M'_1} : (\; ParteFísica_{X_1} \,.\, Etiqueta_{X_1}\,,\, \cdots\,) \\
\{\; ParteFísica_P \,.\, Etiqueta_P : ParteLógica_P \}\,\&\, \ldots\,]\!]
\end{array}\right\} 29. \\
\qquad] \\
)
\end{array}
$$

26. Traducción del envío de mensaje mediante la regla 4.27 de la página 97.
27. Traducción de un programa a través de la regla 4.13 en la página 93.
28. Traducción de una Lista de restricciones con la regla 4.25 en la pág. 96.
29. Traducción de una Lista de métodos utilizando la regla 4.34 en la página 99.

Figura 5.22: Resolución de la traducción de la especificación del constructor comunicación. (continuación c)

$= ($

$\underbrace{\mathcal{T}[\![\ ParteFísica_{I'}\ .\ Etiqueta_{I'}\ ]\!]}_{30.}$

$\triangleleft$

$\underbrace{\mathcal{T}[\![\ Etiqueta_M\ :\ [\ (\ P.Fís_{A_1}\ .\ Etiqueta_{A_1}\ )\ P.Fís_{K_1}\ .\ Etiqueta_{K_1}\ ,\ \ldots\ ]\ ]\!]}_{31.}$

then $ProgramaPiCO\_Q$

$|$

$(\ RestriccionesRecepción_{\phi}\ ,\ RestriccionesDelegación_{\delta}\ )$

$\triangleright\ [$

$\left.\begin{array}{l}\mathcal{T}[\![\ Etiqueta_{M_1'}\ :\ (\ ParteFísica_{X_1}\ .\ Etiqueta_{X_1}\ ,\ \cdots\ )\\ \qquad \{\ ParteFísica_P\ .\ Etiqueta_P\ :\ ParteLógica_P\ \}\ ]\!]\end{array}\right\}$ 32.

$\&$

$\vdots$

$]$

$)$

30. Traducción de Identificador mediante la regla 4.20 de la página 95.
31. Traducción de un mensaje a través de la regla 4.28 en la página 97.
32. Traducción de un método en objeto empleando la regla 4.35 en la página 99.

$I'\ \triangleleft\ l\ :\ [\ \underbrace{\mathcal{T}[\![\ (\ P.Física_{A_1}\ .\ Etiqueta_{A_1}\ )\ P.Física_{K_1}\ .\ Etiqueta_{K_1}\ ,\ \ldots\ ]\!]}_{33.}\ ]$

then Q

$|$

$(\ \phi_{sender}, \delta_{forward}\ ) \triangleright\ [\ l\ :\ (\ \underbrace{\mathcal{T}[\![\ ParteFísica_{X_1}\ .\ Etiqueta_{X_1}\ ,\ \cdots\ ]\!]}_{34.}\ )$

$\underbrace{\mathcal{T}[\![\{\ ParteFísica_P\ .\ Etiqueta_P\ :\ ParteLógica_P\ \}]\!]}_{35.}$

$\&\ \cdots\ ]$

$)$

Figura 5.23: Resolución de la traducción de la especificación del constructor comunicación. (continuación d)

33. Traducción de una lista de Aplicaciones mediante la regla 4.29 de la pág. 98.
34. Traducción de una lista de Identificadores por la regla 4.19 en la página 95.
35. Traducción de un programa empleando la regla 4.13 en la página 93.

$$
\begin{array}{l}
= (\\
\quad I' \triangleleft l : [\\
\qquad \underbrace{\mathcal{T}[\![(ParteFísica_{A_1} . Etiqueta_{A_1}) ParteFísica_{K_1} . Etiqueta_{K_1}]\!]}_{36.} \\
\qquad , \\
\qquad \vdots \\
\qquad , \\
\qquad \underbrace{\mathcal{T}[\![(ParteFísica_{A_n} . Etiqueta_{A_n}) \; ParteFísica_{K_n} . Etiqueta_{K_n}]\!]}_{36.} \\
\qquad] \textbf{ then } Q \\
\quad | \\
\quad (\phi_{sender}, \delta_{forward}) \rhd [\\
\qquad\qquad l : (\\
\qquad\qquad\qquad \underbrace{\mathcal{T}[\![ParteFísica_{X_1} . Etiqueta_{X_1}]\!]}_{37.} \\
\qquad\qquad\qquad , \\
\qquad\qquad\qquad \vdots \\
\qquad\qquad\qquad , \\
\qquad\qquad\qquad \underbrace{\mathcal{T}[\![ParteFísica_{X_n} . Etiqueta_{X_n}]\!]}_{37.} \\
\qquad\qquad) \; ProgramaPiCO\_P \\
\qquad\quad \& \\
\qquad\quad \vdots \\
\qquad] \\
\quad)
\end{array}
$$

Figura 5.24: Resolución de la traducción de la especificación del constructor comunicación. (continuación e)

36. Traducción de una aplicación mediante la regla 4.30 de la página 98.

37. Traducción de un Identificador a través de la regla 4.20 en la página 95.

$$= (\; I' \lhd l : [\underbrace{\mathcal{T} [\![P.Fís_{K_1} . Etiqueta_{K_1}]\!]}_{38.} , \cdots , \underbrace{\mathcal{T} [\![P.Fís_{K_n} . Etiqueta_{K_n}]\!]}_{38.}] \; \textbf{then} \; Q \; | \; (\phi_{sender}, \delta_{forward}) \rhd [l : (\underbrace{Etiqueta_{X_1}}_{39.} , \cdots , \underbrace{Etiqueta_{X_n}}_{39.}) P \; \& \; \cdots] \;)$$

38. Traducción de Identificador mediante la regla 4.20 de la página 95 y asignación de etiqueta impuesta por el programador.

39. Asignación de etiqueta impuesta por el programador.

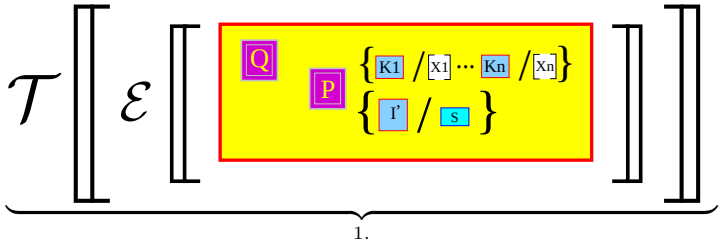
$$= (\; I' \lhd l : [\underbrace{K_1 , \cdots , K_n}_{40.}] \; \textbf{then} \; Q \; | \; (\phi_{sender}, \delta_{forward}) \rhd [l : (\underbrace{x_1 , \cdots , x_n}_{40.}) P \; \& \; \cdots]$$

40. Empleando $\widetilde{y} \stackrel{\text{def}}{=} y_1, y_2, \cdots, y_n$.

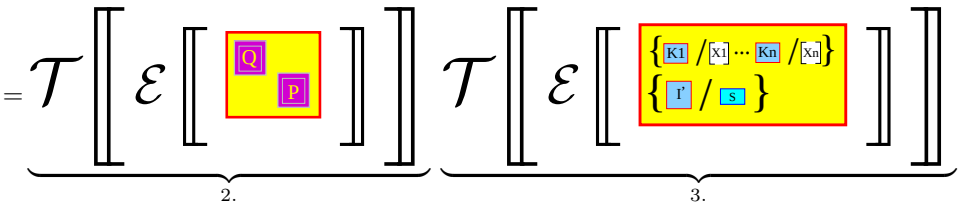
$$= (\; I' \lhd l : [\widetilde{K}] \; \textbf{then} \; Q \; | \; (\phi_{sender}, \delta_{forward}) \rhd [l : (\widetilde{x}) P \; \& \; \cdots] \;)$$

Figura 5.25: Resolución de la traducción de la especificación del constructor comunicación. (continuación f)

III. De igual forma, se procede con la traducción de la especificación de los procesos concurrentes GraPiCO etiquetados como Q y P, los cuales comprenden la parte derecha de la premisa 2 presentada en la Figura 5.18 de la página 120, así:



1. Resolviendo aparte la sentencia sustitución y el constructor a modificar.



2. Especificación de concurrencia por la fórmula en la fig. 3.8 de la pág. 53.
3. Especificación de cada constructor en la sentencia que representa la sustitución.

$$= \mathcal{T}[\![(\underbrace{\mathcal{E}[\![Q]\!]}_{4.} \mid \underbrace{\mathcal{E}[\![P]\!]}_{4.})]\!] \begin{array}{l} \{\, \mathcal{T}[\![\underbrace{\mathcal{E}[\![K_1]\!]}_{5.}]\!] \,/\, \mathcal{T}[\![\underbrace{\mathcal{E}[\![x_1]\!]}_{5.}]\!], \cdots, \mathcal{T}[\![\underbrace{\mathcal{E}[\![K_n]\!]}_{5.}]\!] \,/\, \mathcal{T}[\![\underbrace{\mathcal{E}[\![x_n]\!]}_{5.}]\!] \,\}, \\ \{\, \mathcal{T}[\![\underbrace{\mathcal{E}[\![I']\!]}_{5.}]\!] \,/\, \mathcal{T}[\![\underbrace{\mathcal{E}[\![S]\!]}_{6.}]\!] \,\} \end{array}$$

4. Especificación de programa a través de la fórmula en la fig. 3.7 de la pág. 53.
5. Especificación de Identificadores por la fórmula en la fig. 3.15 de la pág. 56.
6. Especificación de receptor empleando la fórmula en la fig. 3.37 de la pág. 65.

$$= \mathcal{T}[\![\, (\, \{\, ParteFísica_Q \,.\, Etiqueta_Q \,:\, ParteLógica_Q \,\} \\ \quad \mid \\ \quad \{\, ParteFísica_P \,.\, Etiqueta_P \,:\, ParteLógica_P \,\}\,)\,]\!] \Big\}\ 7.$$

$$\{\, \underbrace{\mathcal{T}[\![\, P.Física_{K_1} \,.\, Etiqueta_{K_1} \,]\!]}_{8.} \,/\, \underbrace{\mathcal{T}[\![\, P.Física_{x_1} \,.\, Etiqueta_{x_1} \,]\!]}_{8.}, \cdots \},$$

$$\{\, \underbrace{\mathcal{T}[\![\, ParteFísica_{I'} \,.\, Etiqueta_{I'} \,]\!]}_{8.} \,/\, \underbrace{\mathcal{T}[\![\, ParteFísica_S \,.\, Name(S) \,]\!]}_{8.} \,\}$$

7. Traducción del operador de concurrencia con la regla 4.14 de la pág. 94.
8. Traducción de Identificadores y recepción por la regla 4.20 de la página 95.

Figura 5.26: Resolución de la traducción de procesos concurrentes en una comunicación.

$$= \Big(\underbrace{\mathcal{T}\,[\![\, \{\, ParteFísica_Q \,.\, Etiqueta_Q \;:\; ParteLógica_Q \,\} \,]\!]}_{9.} \Big| \underbrace{\mathcal{T}\,[\![\, \{\, ParteFísica_P \,.\, Etiqueta_P \;:\; ParteLógica_P \,\} \,]\!]}_{9.} \Big) \underbrace{\{K_1/x_1\,, \cdots,\, K_n/x_n\},\ \{I'/\textbf{\textit{sender}}\}}_{10.}$$

9. Traducción de programas empleando la regla 4.13 de la página 93.

10. Empleando $\widetilde{y} \stackrel{\text{def}}{=} y_1, y_2, \cdots, y_n$ y utilizando una única lista de sustituciones.

$$= \Big(\underbrace{\mathcal{T}\,[\![\, ParteLógica_Q \,]\!]}_{11.} \Big| \underbrace{\mathcal{T}\,[\![\, ParteLógica_P \,]\!]}_{11.} \Big) \{\tilde{K}/\tilde{x}\,, I'/\textbf{\textit{sender}}\}$$

11. Traducción de procesos con 4.13 de la pág. 93 y que el método l pertece a P.

$$= (\, Q \,|\, P\,\{\tilde{K}/\tilde{x}\,, I'/\textbf{\textit{sender}}\}\,)$$

Figura 5.27: Resolución de la traducción de la especificación de procesos concurrentes resultantes de una comunicación (continuación).

IV. Luego de la presentación de las premisas en I y por los pasos II y III se construye el diagrama conmutativo de la Figura 5.28 que muestra, en efecto, cómo los procesos de especificación $\mathcal{E}$ y traducción $\mathcal{T}$ conservan las relaciones de reducción de los procesos de envío de mensajes.

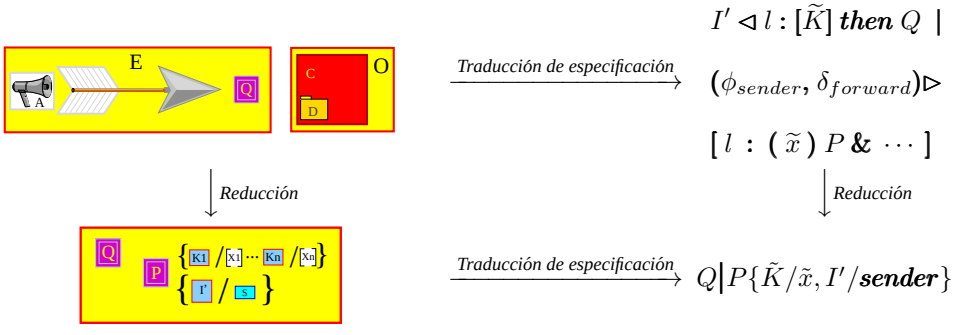


Figura 5.28: Diagrama de la corrección de las reglas de traducción de comunicación.

□

Corrección del constructor de delegación.

I. Para la demostración de la conservación de la semántica de la traducción del constructor de delegación de mensajes entre objetos las premisas requeridas se presentan en la Figura 5.29 y son explicadas a continuación:

1. Premisa.

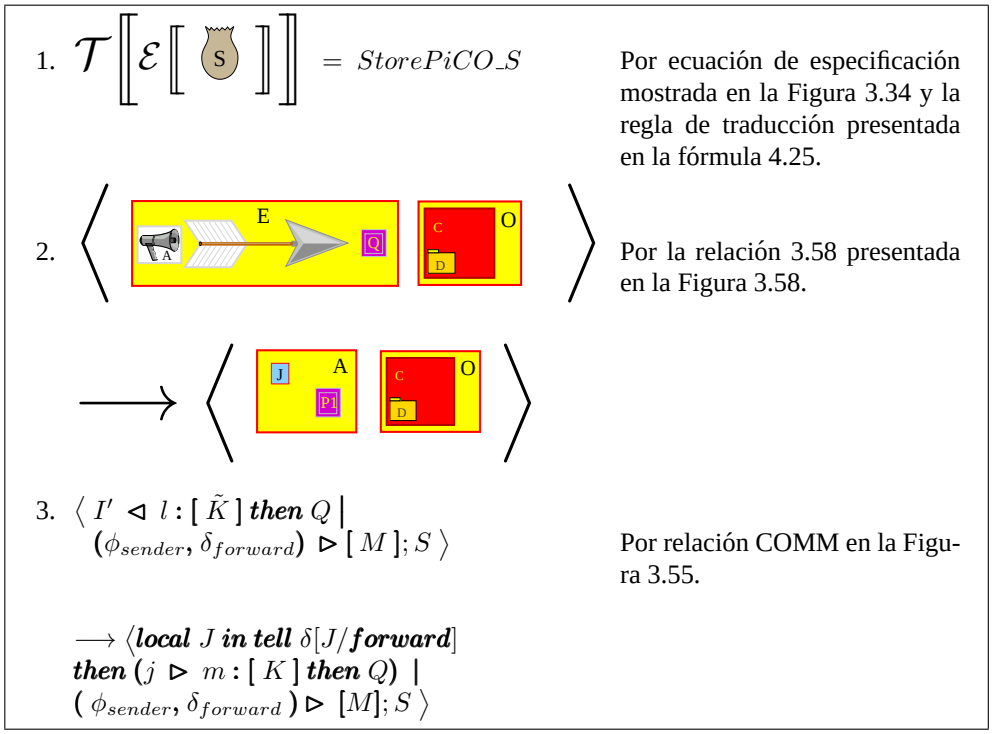
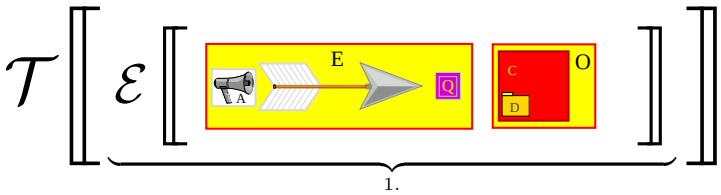


Figura 5.29: Premisas para el proceso de traducción.

II. Posterior a la presentación de las premisas se procede con la traducción de la especificación de la delegación de mensaje entre objetos, así:



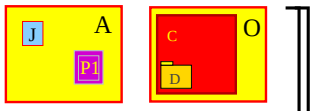
1. Empleando la resolución del constructor comunicación de la Figura 5.19 en la página 121.

$$= (\ I' \lhd l : [\ \widetilde{K}\ ]\ \textbf{\textit{then}}\ Q \mid (\ \phi_{sender}, \delta_{forward}\ ) \rhd [\ M\ ])$$

Figura 5.30: Resolución de la traducción de la especificación del constructor delegación.

III. De igual forma se procede con la traducción de la especificación de la creación del nuevo ámbito para la variable J, lo cual comprende la parte derecha de la premisa 2 presentada en la Figura 5.29 de la página 130, así:

$$\mathcal{T}\Big[\!\Big[\underbrace{\mathcal{E}\Big[\!\Big[}_{1.}$$



$$\underbrace{\Big]\!\Big]}_{1.}\Big]\!\Big]$$

1. Especificación del operador de concurrencia por la fórmula en la fig. 3.8 de la pág. 53 y la sustitución $[J/\boldsymbol{forward}]$ en la relación 3.58 de la pág. 80.

$$= \mathcal{T}[\![\,(\,\underbrace{\mathcal{E}[\![\,A\,]\!]}_{2.}[J/\boldsymbol{forward}\,]\;|\;\underbrace{\mathcal{E}[\![\,O\,]\!]}_{3.}\,)\,]\!]$$

2. Resolución de creación ámbito presentado en la fig. 5.3 de la pág. 109.

3. Especificación de objeto por la fórmula en la fig. 3.16 de la pág. 56.

$$= \boldsymbol{local}\ Id.PiCO\_J\ \boldsymbol{in}\ \underbrace{\mathcal{T}[\![\,\mathcal{E}[\![\,P_1{}^a\,]\!]\,]\!]}_{4.}[J/\boldsymbol{forward}\,]\;|\;(\,\phi_{sender},\delta_{forward}\,)\rhd[\,M\,])$$

4. Con la relación de transición de delegación en la fig. 3.58 en la pág. 80 y la resolución de imposición de restricciones en la fig. 5.11 de la pág. 115.

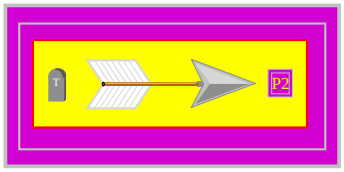
$$= \boldsymbol{local}\ Id.PiCO\_J\ \boldsymbol{in\ tell}\ \delta[J/\boldsymbol{forward}\,]\ \boldsymbol{then}\ \underbrace{\mathcal{T}[\![\,\mathcal{E}[\![\,P_2{}^b\,]\!]\,]\!]}_{5.}\;|\;(\,\phi_{s.},\delta_{f.}\,)\rhd[\,M\,])$$

5. Con la relación de transición de delegación en la fig. 3.58 en la pág. 80 y la resolución de comunicación en la fig. 5.19 de la pág. 121.

$$= \boldsymbol{local}\ Id.PiCO\_J\ \boldsymbol{intell}\ \delta[J/\boldsymbol{forward}\,]\ \boldsymbol{then}\ J\rhd m:[\,k]\ \boldsymbol{then}\ Q\;|\;(\phi_{s.},\delta_{f.})\rhd[M])$$

---

[a] $Ex(P_1) =$



[b] $Ex(P_2) =$

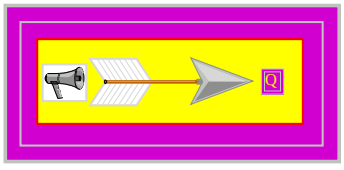


Figura 5.31: Resolución de la traducción de la especificación de la creación del nuevo ámbito para la variable J.

IV. Luego de la presentación de las premisas en I y por los pasos II y III se construye el diagrama conmutativo de la Figura 5.32 que muestra, en efecto, cómo los procesos de especificación $\mathcal{E}$ y traducción $\mathcal{T}$ conservan las relaciones de reducción de los procesos de delegación de mensajes entre objetos.

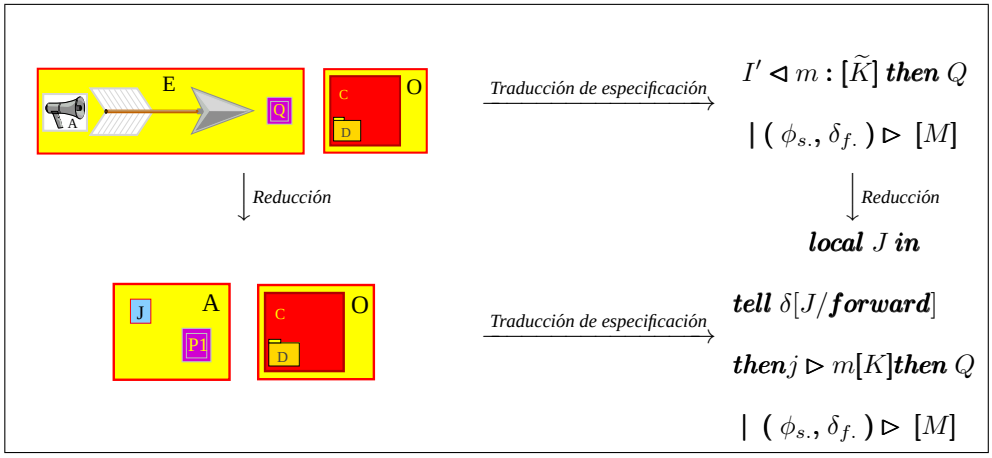


Figura 5.32: Diagrama conmutativo de la corrección de las reglas de traducción del constructor visual de delegación.

□

De esta manera, con la demostración de la conservación de la semántica entre los constructores visuales de GraPiCO y los constructores textuales del cálculo PiCO, se puede afirmar que los programas editados visualmente con los contructores GraPiCO tienen la misma semántica de los programas obtenidos después de la utilización de las funciones de especificación $\mathcal{E}$ y traducción $\mathcal{T}$.

Capítulo 6

Comprobación sintáctica de la especificación textual de lenguajes visuales mediante símbolos de sincronización

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:

1. Análisis léxico.
2. Análisis sintáctico.
3. Análisis semántico.
4. Optimización.
5. Traducción.

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 6.1.

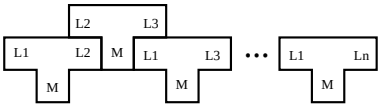


Figura 6.1: Compilador escalonado.

[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.

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 [CP01], 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 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.

6.1. Alternativa para comprobación sintáctica de lenguajes visuales

Independiente de la especificación gramatical que se utilice, la comprobación sintáctica tiene dos niveles:

1. Sintaxis independiente del contexto.
2. Sintaxis dependiente del contexto.

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. 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 sintaxis 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.

Antes de continuar se necesita una definición.

Definición 11. *El conjunto* **PRIMEROS_ÍCONOS** *consta de los primeros íconos que se puedan generar de una producción.*

$$\textit{PRIMEROS\_ÍCONOS}(X) = \begin{cases} imagen_X \\ \qquad \textit{Si } X \to S'\ Sd_1\ (X - Y)\ imagen_X\ \sim\ \ldots, \\ \textit{PRIMEROS\_ÍCONOS}(PL_X{}^a) \\ \qquad \textit{Si } X \to S'\ Sd_1\ \epsilon\ \sim\ Et_X{}^b\ :\ PL_X\ Sd_2, \\ \bigcup \textit{PRIMEROS\_ÍCONOS}(X_i) \\ \qquad \textit{Si } X \to S'\ Sd_1\ X_1\ \widehat{Sc_1}\ \cdots\ \widehat{Sc_{n-1}}\ X_n\ Sd_2, \\ \textit{PRIMEROS\_ÍCONOS}(Y \veebar Z)^c \\ \qquad \textit{Si } X \to Y \mid Z. \end{cases} \tag{6.1}$$

[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:

$$X \to S'\ Sd_1\ ParteFísica_X\ \sim\ Etiqueta_X\ :\ \underbrace{ParteLógica_X}_{LX}\ Sd_2 \tag{6.2}$$

  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 $expansión$:

$$LX \rightarrow S' \; Sd_1 \; X_1 \; \widehat{Sc_1} \; \cdots \; \widehat{Sc_{n-1}} \; X_n \; Sd_2 \tag{6.3}$$

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 12. *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 6.2).*

Definición 13. *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 6.2).*

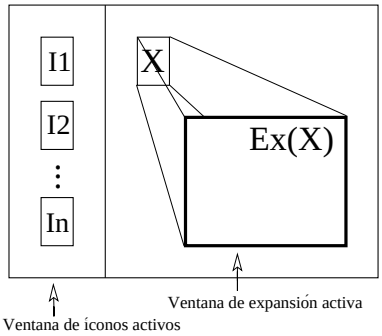


Figura 6.2: Ventana de constructores activos y ventana de expansión activa.

Consecuente con las definiciones 12 y 13 se puede definir qué constructores pueden intervenir en la parte lógica de cada constructor visual expresado con una Gsig.

Definición 14. *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).

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

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

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

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

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

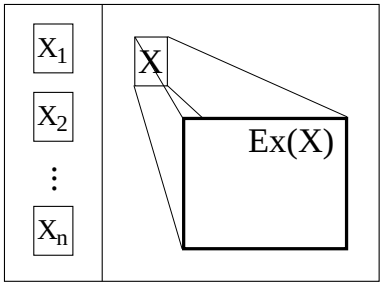


Figura 6.3: Íconos permitidos en la ventana de expansión activa del ícono del constructor X.

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. Y poder 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 palabras, 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[3]).

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 sintaxis independiente del contexto. Lo anterior se consigue empleando el mecanismo de traducción T_Gsig presentado en el capítulo 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) y métodos (lista de parámetros), para recorrer cada una de las tablas y verificar las condiciones dependientes del contexto como: todas las variables deben tener un valor asociado, la cantidad de argumentos en un método debe 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.

6.1.1. Comentarios sobre el mecanismo de análisis sintáctico planteado

De esta manera, mediante el mecanismo de $dividir\ y\ conquistar$ se ha reducido la complejidad 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 sintácticas 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.

6.2. Implementación de GraPiCO

La implentación final de lo que comprende la investigación mostrada en el presente documento se dividió en dos proyectos.

- Editor de programas visuales GraPiCO: Programa que permite la edición de programas visuales y su correspondiente traducción hacia un lenguaje intermedio de almacenamiento. Para lograr la implementación de este desarrollo se dirigió el proyecto de grado títulado: ”Desarrollo del entorno gráfico para el cálculo visual GraPiCO”del estudiante Flóver Sánchez Ortega de la Universidad del Valle.

- Traductor de lenguaje intermedio hacia cálculo computacional: De igual forma, para conseguir la traducción del código intermedio de almacenamiento hacia el cálculo PiCO con su respectivo análisis sintáctico de las características independientes del contexto y el análisis semático mediante Tablas Hash, se dirigió el proyecto de grado títulado ”Diseño e implementación de la traducción desde la especificación textual del

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

cálculo visual GraPiCO hacia una representación semánticamente válida en cálculo Pi-CO"de los estudiantes Johanna Noguera León y Alejandro Bermúdez Martínez de la Universidad del Valle.

Los dos proyectos fueron dirigidos por el autor de la investigación. Posterior a la presentación de los dos proyectos de grado, el autor efectuó la pueba conjunta de los dos programas resultantes de los proyectos.

Capítulo 7

Breve discusión de las ventajas de los lenguajes visuales y el cálculo GraPiCO

En esta parte del documento se discutirán las ventajas del cálculo visual GraPiCO frente a su antecesor el cálculo PiCO. Primero, se presenta un estudio comparativo entre los lenguajes de programación textuales y visuales; luego, se discuten las características más representativas del cálculo GraPiCO y se comparan con las del cálculo PiCO.

El proyecto de investigación *Efficiency of the Visual* (University of Texas at Dallas) del profesor Mihai Nadin, presentó en [Nad04] un estudio comparativo entre lenguajes de programación. El estudio arrojó los resultados presentados en la tabla 7.1.

Criterio\lenguaje	Visual (Labview)	Textuales(C, C++, Java)
Cantidad de Código	Poco	Aceptable
Implementación	Flexible	Demorado
Interacción	Intuitivo	Requiere análisis
Nivel de Abstracción	Alto	Limitado
Mec. de Abstracción	Limitado	Alto
Aprendizaje	Anima a participar y aprender	Requiere entrenamiento y libros

Tabla 7.1: Tabla de resultados de estudio comparativo.

En el informe del profesor Nadin se discuten las siguientes ideas con respecto a los Entornos de los lenguajes de programación visual frente a los de programación textual:

- Complejidad espacial: El tamaño de código (cantidad de constructores del lenguaje) es menor que en uno textual, porque se aprovechan las ventajas expresivas de los lenguajes visuales.
- Implementación: De igual forma, el trabajo de implementación de programas es una experiencia más eficiente porque los visuales sacan beneficio de los entornos gráficos

de programación, los que han presentado grandes avances gracias al apoyo de la industria que requiere atender un mercado donde, cada vez más, se difunde e impone la utilización de los multimedios.

- Interacción: También brinda al programador una interfaz más intuitiva que uno textual, debido al acercamiento de los elementos visuales a las imágenes creadas en la mente cuando se piensa en la solución de un problema.
- Niveles de abstracción: El informe dice que son más altos y potentes que en los lenguajes de programación textuales, dada la sintaxis lineal y secuencial de estos últimos.
- Mecanismos de abstracción: Su eficiencia limita los Ambientes de programación visuales en razón de que estos parten desde la perspectiva del usuario, mientras que los textuales inician desde la perspectiva del sistema de cómputo.
- Aprendizaje: Finalmente, invita a los usuarios a participar y a aprender. Dado que para la utilización de los lenguajes textuales el usuario requiere transformar las construcciones visuales creadas en su pensamiento en cadenas de caracteres.

Y arrojó la siguiente conclusión:

"Los lenguajes de programación visual intentan cambiar la experiencia de programar desde la perspectiva del computador hacia la perspectiva del usuario. En pocas palabras, La programación visual es el poder de la percepción visual en humanos extendida al proceso de la programación. La programación visual simplifica el proceso de programación significativamente y aumenta la interacción humano-computador al ofrecer una interfaz más lógica e intuitiva. En esencia, la programación visual es una instancia comprensiva del caso de la experiencia visual de humanos manipulando innovaciones tecnológicas."

Aunque tiene la misma complejidad aprender a usar un lenguaje de programación visual como otro textual, ambos con equivalencia expresiva, el estudio muestra varias ventajas que hacen muy atractivos a los primeros; unas ligadas directamente a la forma como los humanos de manera inherente construyen dibujos con su cerebro al diseñar un resultado y otras, relacionadas con tendencias y costumbres adquiridas por el surgimiento de nuevas tecnologías que permiten desarrollos más exigentes como los multimedios.

Luego de mostrar las conclusiones del estudio de las ventajas de los lenguajes visuales frente a los lenguajes textuales, se presentará una comparación entre GraPiCO y PiCO.

7.1. Expresividad del cálculo visual GraPiCO

En cuanto a la expresividad de las ventajas de GraPiCO frente a PiCO para su discusión se empleará un programa a manera de corpus[1] muy concreto, pues la intención del documento no es la de elaborar un corpus paralelo[2] para efectuar lingüística de corpus[3] sino la de mostrar que, en efecto, están presentes las características que motivaron la investigación: Mejorar la

[1] Actos reales de un lenguaje (como programas editados) que describen los sistemas lingüísticos.

[2] Tipo de corpus que se caracteriza por presentar una correspondencia uno a uno entre programas originales en un lenguaje y su traducción a otro lenguaje para analizar equivalencias.

[3] Rama de la lingüística centrada en el estudio del uso de un lenguaje tomando como objeto de estudio actos reales del lenguaje, como programas editados.

agilidad para crear y modificar programas; facilitar la modelación de los gráficos creados de forma innata por las personas al idear soluciones; y permitir que el nivel de abstracción sea elegido por el usuario. Todo bajo la propiedad de la conservación de la semántica, concepto ya demostrado en el capítulo 5.

Antes de discutir las características se muestra un programa donde son utilizados todos los contructores del cálculo PiCO y posteriormente se presenta su equivalente en GraPiCO.

7.1.1. Programa PiCO: salón de clase

El programa modela un salón de clase donde están presentes un profesor y un grupo de Estudiantes (particionados en los conjuntos "egoístas" y "generosos"). El profesor requiere un esfero; para obtenerlo, pide prestado uno con tinta azul en voz alta a la clase. Para modelar esta situación se programaron cuatro procesos concurrentes explicados a continuación (primero, presentando cada componente del proceso en términos computacionales con su respectiva letra como índice para cada segmento de código y después, una explicación con conceptos del mundo real –*en letra itálica* –):

1. Alumno "egoísta". proceso que consiste en la creación de un nuevo ámbito para los métodos $cargar$ y set dentro del programa compuesto por cuatro procesos concurrentes identificados con las letras de la a) a la d) en la Figura 7.1:

   a) Objeto prestador de esferos: Tiene como restricción de recepción la aceptación de los mensajes enviados por una instancia igual al número 20; objeto compuesto también, por la restricción de delegación expresando que los mensajes sin posibilidad de ser contestados deberán ser enviados por una instancia igual al número 30; finalmente, se observa por el espacio entre los corchetes que este objeto no tiene método alguno.

   Esta línea de código nos modela la situación donde el Estudiante escucha a personas que estén a 20 metros del tablero (el profesor) y cuando no puede contestar una petición siempre expresa que le pregunten a las personas ubicadas a 30 metros del tablero (Los Estudiantes "generosos"), este Estudiante es denominado "egoísta" porque no presta sus útiles.

   b) c) Mensaje cargador de esferos: Un proceso implicación que cuenta como antecedente el envío del mensaje $cargar$ (el primer argumento del método representa $color$, 1 significa azul, 2 rojo. El segundo argumento representa el $estado$, 1 presente, 0 prestado) y como consecuente, un proceso Nulo.

   Con esta línea se modela la introducción de un esfero (de un determinado color) en la Cartuchera. La introducción del esfero a la Cartuchera no genera acción posterior alguna.

   d) Objeto Cartuchera: Cuenta como restricción de recepción la aceptación de los mensajes enviados por una instancia igual al número 0; no cuenta con restricciones de delegación y tiene un único método denominado $cargar$.

- Método $cargar$: Consiste en la creación de un objeto a partir de los Parámetros $color$ y $estado$, de manera que este último tenga como Condición de recepción la aceptación de los mensajes enviados por una instancia igual al valor del Parámetro $color$ y que contiene los métodos get y set.
  - Método get: función que se encarga de imponer el Valor de $estado$ en la variable z; si este procedimiento se puede efectuar, se procede al envío del mensaje $cargar$ para la activación de un nuevo objeto con los mismos Valores actuales de $color$ y $estado$.
  - Método set: Se encarga de Enviar el mensaje $cargar$ para la creación de un nuevo objeto con los Valores $color$ y $estado\_new$.

$$
\begin{array}{l}
\textbf{\textit{clone local}}\ cargar, set\ \textbf{\textit{in}} \\
\quad (\\
a)\quad (\textbf{\textit{sender}} = 20,\ \textbf{\textit{forward}} = 30) \triangleright [\,] \\
b)\ |\ 0 \lhd cargar : [\,1, 1]\ \textbf{\textit{then}}\ \boldsymbol{O} \\
c)\ |\ 0 \lhd cargar : [\,2, 1]\ \textbf{\textit{then}}\ \boldsymbol{O} \\
d)\ |\ (\textbf{\textit{sender}} = 0) \triangleright [\ cargar : (\ color, estado\ )\ (\textbf{\textit{sender}} = color) \triangleright \\
\qquad\qquad\qquad\qquad [\\
\qquad\qquad\qquad\qquad\ get : (\ z\ )\ \textbf{\textit{tell}}\ z = estado\ \textbf{\textit{then}} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad 0 \lhd cargar : [\ color, estado]\ \textbf{\textit{then}}\ \boldsymbol{O} \\
\qquad\qquad\qquad\qquad\ \&\ \\
\qquad\qquad\qquad\qquad\ set : (\ estado\_New\ ) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad 0 \lhd cargar : [\ color, estado\_New]\ \textbf{\textit{then}}\ \boldsymbol{O} \\
\qquad\qquad\qquad\qquad] \\
\qquad\qquad\qquad] \\
\quad)
\end{array}
$$

Figura 7.1: Programa salón de clase (definición de alumno “egoísta”).

El segmento de código $d)$ modela el artículo Cartuchera donde se pueden introducir los esferos, los cuales se pueden revisar si están presentes en la Cartuchera o están prestados (mediante el método get), y en caso de que se quiera, se pueden prestar en un momento dado (a través del método set); en cualquiera de las situaciones, cada vez, se crea un nuevo esfero con las características adecuadas de color y estado.

2. Alumno “generoso”. El código de este proceso se encuentra en la Figura 7.2.

La diferencia entre los procesos alumno “egoísta” y alumno “generoso” radica en el objeto prestador de esferos codificado en la línea etiquetada con e).

- Objeto prestador de esferos: Es el encargado de atender las solicitudes de préstamo de esferos, tiene como restricción de recepción la aceptación de los mensajes enviados por una instancia igual al número 30; ya que no cuenta con restricción de delegación si este objeto no tiene el método pedido no envía el mensaje de nuevo; de igual forma, el objeto prestador de esferos tiene el método $prestar\_esfero$.
  - Método $prestar\_esfero$: Se encarga de recibir la variable $color$ y hace que esta misma envíe el mensaje set con el número 0 como Parámetro. Posterior a su ejecución no se activa procedimiento alguno.

La diferencia entre un alumno “generoso” y uno “egoísta” es que:

- *El “generoso” en efecto presta sus útiles (en este caso, los únicos útiles son dos esferos, uno rojo y otro azul).*
- *También el alumno “generoso” no hace que las peticiones que no pueda contestar sean efectuadas a otra persona.*

```
| clone local cargar, set in
    (
e)  (sender = 30) ▷ [ prestar_esfero : ( color ) color ◁ set : ( 0 ) then O ]
    | 0 ◁ cargar : [ 1, 1] then O
    | 0 ◁ cargar : [ 2, 1] then O
    | (sender = 0) ▷ [ cargar : ( color, estado ) (sender = color) ▷
                                [
                                  get : ( z ) tell z = estado then
                                                        0 ◁ cargar : [ color, estado] then O
                                  &
                                  set : ( estado_New )
                                                        0 ◁ cargar : [ color, estado_New] then O
                                ]
                      ]
    )
```

Figura 7.2: Programa salón de clase (definición de alumno “generoso”).

3. Profesor. Proceso presentado en la línea $f)$ de la Figura 7.3 que consiste en la creación de un nuevo ámbito para el método $pedir\_esfero$ dentro del programa compuesto por un objeto, el cual tiene como restricción de recepción la aceptación de los mensajes enviados por una instancia igual al número 0; no cuenta con restricciones de delegación y tiene un único método denominado $pedir\_esfero$.

    - Método $pedir\_esfero$: Se encarga de hacer que el número 20 realice la petición del método $prestar\_esfero$ con el argumento $color$, la ejecución de este método no genera proceso posterior alguno.

    La línea de código etiquetada con $f)$ en la Figura 7.3 modela la situación en la que un profesor requiere un esfero de un determinado color, el cual solicita a los Estudiantes mediante la frase "necesito que alguien me preste un esfero color X": Si el esfero le es prestado o no, no produce después algún evento.

4. Verificación y petición de esfero. Proceso presentado en la línea $g)$ de la Figura 7.3, que consiste en una implicación con las siguientes partes:

    - Antecedente: El número cero (0) efectúa la petición del mensaje get.
    - Consecuente: Una implicación con las siguientes partes:
        - Antecedente: consulta de la restricción de igualdad entre el Identificador e y el número 1.
        - Consecuente: implicación que consiste en la petición del mensaje $pedir\_esfero$ por parte del número cero (0). Esta misma última implicación no tiene consecuente.

    La línea de código etiquetada con $g)$ en la Figura 7.3 presenta la modelación en la que el profesor verifica si algún esfero está disponible, en caso de que así sea, procede a la petición del esfero de un determinado color.

$$
\begin{aligned}
&f) \mid \textbf{\textit{clone local}}\ pedir\_esfero\ \textbf{\textit{in}} \\
&\quad \big((sender = 0)\ \triangleright [pedir\_esfero : (\ color)\ 20 \triangleleft\ prestar\_esfero : [\ color]\ \textbf{\textit{then}}\ \boldsymbol{O}\ ]\big) \\
&g) \mid 0\ \triangleleft\ get : [\ e]\ \textbf{\textit{then ask}}\ e = 1\ \textbf{\textit{then}}\ 0 \triangleleft\ pedir\_esfero : [\ 1]\ \textbf{\textit{then}}\ \boldsymbol{O}.
\end{aligned}
$$

Figura 7.3: Programa salón de clase (definiciones de profesor, y verificación y petición de esfero).

Luego de la presentación del programa PiCO salón de clase, a continuación se muestra una visión de su equivalente en el cálculo visual GraPiCO.

7.1.2. Programa GraPiCO: salón de clase

Como se discutió, el programa modela un salón de clase donde están presentes tres tipos de actores:

- Profesor: Cuenta con el método $Pedir\_esfero$.
- Alumno “egoísta”: Sin método alguno.
- Alumno “generoso”: Contiene los métodos: $Prestar\_esferos$ y $Cargar$.

  Para modelar esto, se construyen con el cálculo visual GraPiCO tres procesos creación de nuevo Ambiente para cada actor respectivamente y un proceso implicación para la verificación de las condiciones iniciales (ver Figura 7.4).

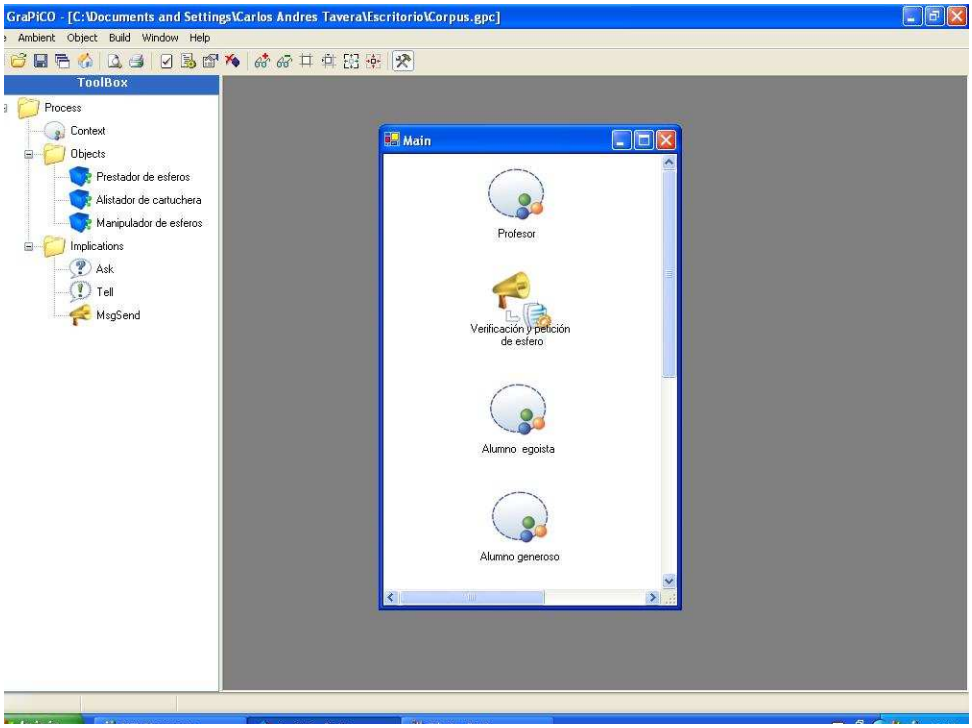


Figura 7.4: Programa salón de clase GraPiCO (nivel de abstracción 1).

Al igual que el diseñó en la sección anterior, para contener los métodos se crearon objetos dentro de cada proceso creación de nuevo Ambiente correspondiente a los actores, así:

- Profesor: Con el objeto $Pedir\_esferos$ el cual tiene el método $Pedir\_esfero$.
- Alumno “generoso”: En el que se crearon dos objetos:
  - Objeto $Prestador\_de\_esferos$: Con el método $Prestar\_esfero$.
  - Objeto $Alistador\_de\_cartuchera$: Con el método $Cargar$.
- Alumno “egoísta”: la única diferencia del proceso creación de nuevo Ambiente modelando un alumno “egoísta” con respecto al alumno “generoso” es que el objeto

$Prestador\_de\_esferos$ no tiene método alguno porque no presta sus esferos por ser "egoísta".

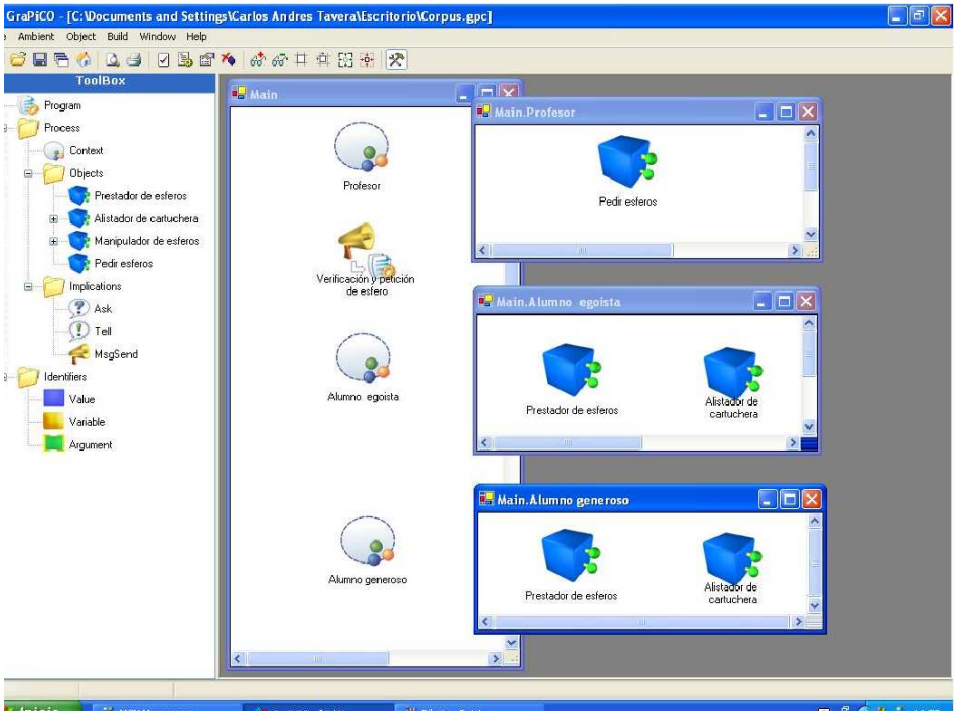


Figura 7.5: Programa salón de clase GraPiCO (nivel de abstracción 2).

La Figura 7.6 muestra que para modelar el comportamiento del objeto $Prestador\_de\_cartuchera$ se diseñó una restricción de recepción $Restriction\ 1$, donde se impone la restricción: $\mathbf{sender} = Color$. Del mismo modo, se observa el método $Cargar$.

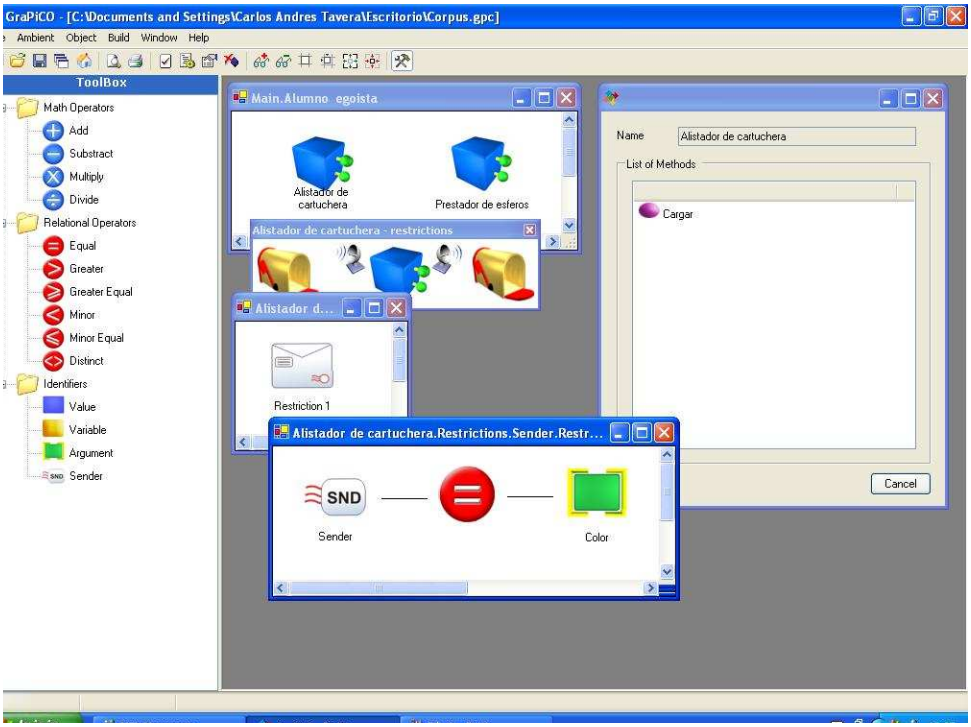


Figura 7.6: Programa salón de clase GraPiCO (más niveles de abstracción).

Con el ejemplo anterior se observan las principales características por las que los lenguajes visuales aventajan a los textuales:

1. Agilidad para crear y modificar programas: Por la herramienta de las ventanas y los menús, el cálculo visual permite la edición de los programas de manera más veloz, pues es más simple el desplazamiento de un ícono a una ventana (la que representa con inmediatez el contexto para el constructor visual movilizado) que escribir todo el código respectivo.

2. Fácil modelación de programas: Dados los íconos y la posibilidad de cambiarlos, las imágenes creadas cuando un programador piensa en la solución a un problema se modelan con mayor coherencia; como las gráficas de los íconos no están asociadas de forma estática, el usuario puede elegir dibujos cada vez más cercanos a los elementos que intervienen.

3. El nivel de abstracción es elegido por el usuario: Gracias a la maximización y minimización de las ventanas, y también por la apertura y el cierre de los íconos (la *expansión*), el usuario puede escoger qué grupos de íconos ver.

Para mostrar la utilidad del cálculo GraPiCO en la modelación de definiciones matemáticas se presentará en la sección 7.2 la modelación de la función factorial.

7.2. Ejemplo final

Luego de mostrar las bondades de la expresividad del cálculo visual GraPiCO, se presentará un último ejemplo de programa visual donde se modela la función clásica para cálculo de factorial mediante recursión de cola:

$$Fac(n,m) = \begin{cases} m & \text{Si } n = 1, \\ Fac(n-1, n*m) & \text{Si } n > 1. \end{cases} \tag{7.1}$$

A continuación, en la Figura 7.7 se presenta la ventana del programa Principal compuesto por un ámbito y dos objetos, correspondientes respectivamente al establecimiento de los valores iniciales de la computación y a los dos casos de ejecución cuando se le calcula el factorial a un número mayor a uno o igual a uno.

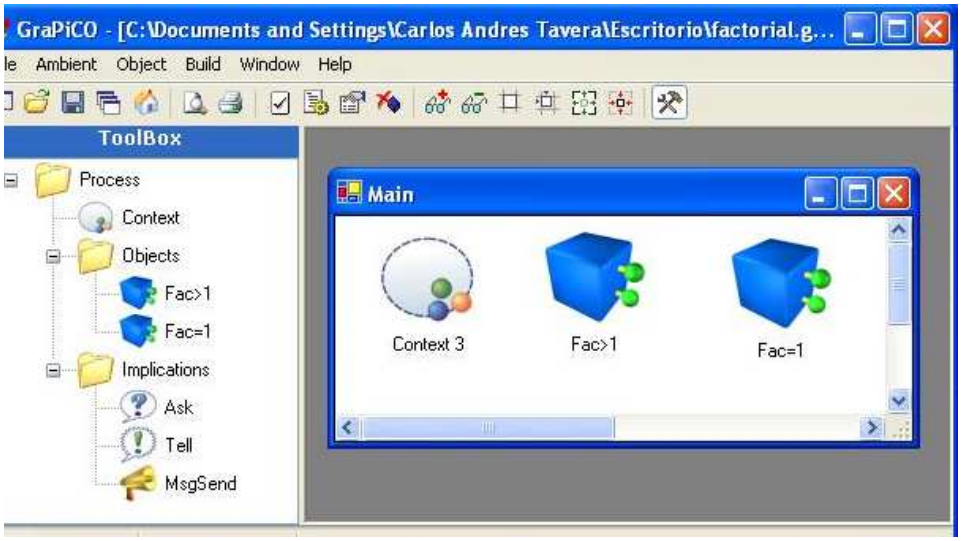


Figura 7.7: Ventana inicial del programa GraPiCO que calcula factorial (Nivel 1).

En la Figura 7.8 se presenta el programa visual con la visión de las ventanas obtenidas luego de la expansión de los íconos de los objetos y el contexto.

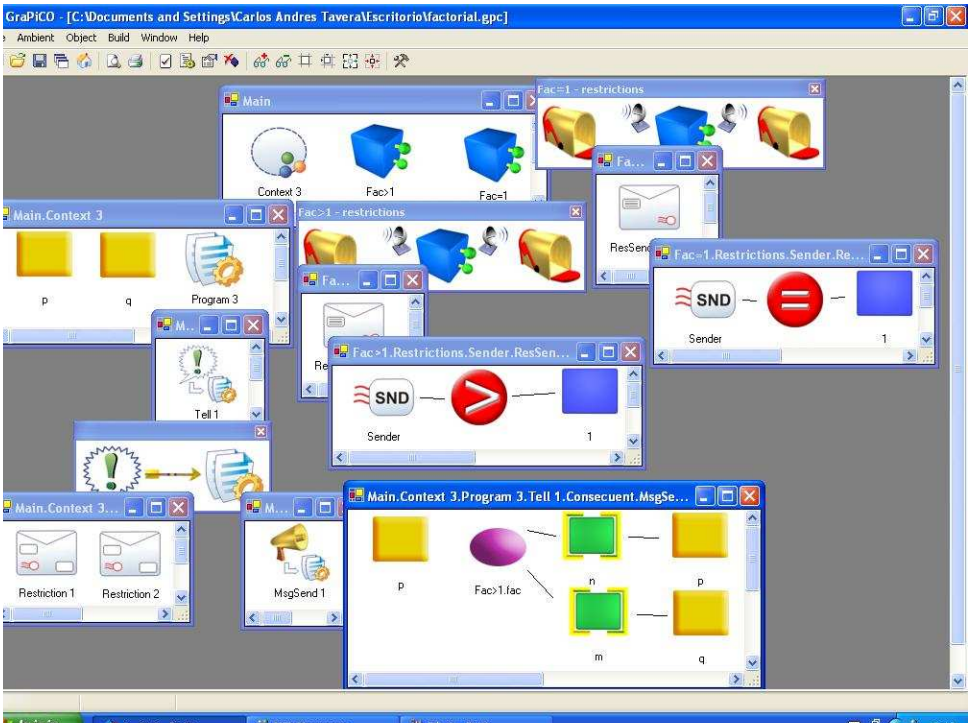


Figura 7.8: Ventana resultado de la expansión de los íconos presentes en la ventana inicial.

En la Figura 7.9 se muestra el resultado de la expansión de los métodos de los objetos y sus respectivos programas.

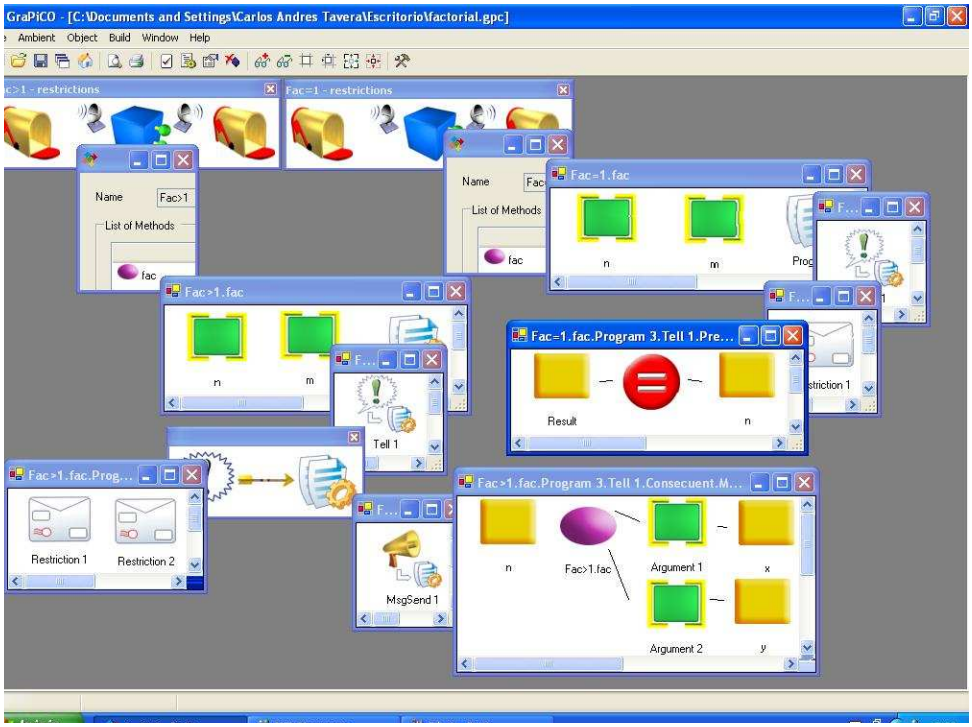


Figura 7.9: Ventana que muestra la expansión de los métodos de los objetos de la ventana inicial.

Se observa cómo la información que no se quiere tener desplegada (en un momento dado no es pertinente) se oculta con la utilización de los niveles de abstracción; e igualmente, se aprecia que la programación visual ofrece un entorno atractivo, amigable y flexible.

En las siguientes secciones se listan los resultados, las conclusiones y el trabajo futuro proyectado con relación al cálculo visual GraPiCO.

PARTE III

EPÍLOGO

Capítulo 8

Resultados, conclusiones y trabajo futuro

8.1. Resultados

El aporte al conocimiento en el área de los lenguajes de programación está representado en los siguientes resultados:

8.1.1. Resultados directos

- GraPiCO: Nuevo cálculo visual, orientado al objeto y concurrente por restricciones, donde la compilación aprovecha las características formales y toda la expresividad sintáctica del cálculo PiCO, ya que se construyó una Correspondencia Uno a Uno entre los constructores PiCO y GraPiCO, y posteriormente se efectuó la demostración de la corrección de la semántica. Este desarrollo contiene:
  - cálculo visual del nuevo cálculo GraPiCO.
  - Prototipo de Editor de cálculo visual GraPiCO.
  - Definición gramatical para la representación textual de lenguajes visuales como el cálculo GraPiCO.
  - Mecanismo de especificación de programas visuales hacia una representación textual.
  - Función de traducción desde programas visuales en representación textual hacia programas textuales sin información visual.
  - Alternativa de comprobación sintáctica para lenguajes visuales.

8.1.2. Resultados indirectos

- Un ambiente de programación visual que realiza el proceso de edición de programas en cálculo visual GraPiCO y la compilación de Código visual GraPiCO a cálculo PiCO.

- Uno de los primeros cálculos de programación visual por restricciones concurrentes Orientado al objeto.
- Un sistema versátil, porque soporta varios paradigmas de lenguajes de programación.
- Un sistema de uso general, dado que puede ser aplicado para modelar problemas inherentes a la programación por restricciones y programación Orientada al objeto (se presentaron un par de ejemplos de programas en las secciones 7.1 y 7.2).

8.2. Conclusiones

1. Dentro de las opciones para crear el nuevo cálculo visual que se requería se eligió la creación de un conjunto de constructores gráficos mediante una Correspondencia Uno a Uno; fue una idea acertada porque asegura (desde el punto de vista léxico) que los cálculos PiCO y GraPiCO tienen el mismo poder expresivo.

2. Tambien se observó que la creación de una nueva forma de especificación gramatical Gsig ayudó a conseguir los objetivos iniciales, porque algunos requerimientos del proyecto no eran posibles de cumplir con los mecanismos presentes durante la ejecución del proyecto. Estás necesidades y sus soluciones son:

   a) Fácil manera de guardar y recuperar los programas visuales. Los desarrollos difundidos hasta el momento están concentrados en la corrección gramatical y dejan de lado la efiencia en la manipulación del almacenamiento o incluso no la contemplan.

   Las Gsig, en cambio, permiten el almacenamiento de forma muy simple, pues su estructura configura un lenguaje regular; de igual manera, como la posición y configuración de cada constructor está contenida y se puede encontrar rápidamente, la recuperación de un programa es directa.

   b) Se asegura que la semántica de los programas PiCO se conserva cuando se programa su equivalente en GraPiCO. Este era otro tema relegado para después de obtener la gramática con los mecanismos existentes, pues los diseños de las sintaxis no planeaban los estudios semánticos.

   En lugar de utilizar los desarrollos presentes las Gsig fueron creadas para posibilitar tratamientos matemáticos, por esta razón cada producción está compuesta de dos partes, una que contiene la información de donde está situado el constructor y qué gráfica está asociada a él (la Parte física), y otra que determina cómo se relaciona éste con los demás (la Parte lógica). Luego, por medio de la construcción del conjunto de reglas de traducción (que permite abstraer la estructura de los programas de su información gráfica y para la que, incluso, le fueron diseñadas sus clases con la intención de fortalecer su implementación) y el método estructural, se consiguió demostrar que los programas resultantes luego de la especificación y

traducción a un código GraPiCO son PiCO. De otro lado, gracias a la elaboración de un conjunto de Reglas de transición del GraPiCO mediante Diagramas Conmutativos fue posible la demostración bien estructurada de la conservación de la semántica de los programas GraPiCO convertidos a PiCO.

c) Brindar la posibilidad de diseñar un conjunto de reglas de especificación para poder transformar un programa visual GraPiCO en un equivalente textual que contenga toda la información. Aunque algunos mecanismos permiten de un programa visual la obtención de uno textual análogo, el resultado es muy complicado o requiere la utilización de software propietario.

   A diferencia de los mecanismos acabados de mencionar, Gsig se apoya en un conjunto de reglas de especificación que convierten cada uno de los constructores visuales en su equivalente textual con toda la información. Estas reglas fueron encaminadas a que cada transformación nos entregue un sintagma de PiCO.

3. Luego de dar solución a los problemas con respecto a la especificación sintáctica, se encuentra que los mecanismos de análisis presentes implican gran esfuerzo porque unos no sirven para tratar el cálculo GraPiCO por dificultades en el poder expresivo de las gramáticas empleadas, y otros, aunque pueden crear una sintaxis para GraPiCO, tienen costos computacionales muy grandes. La solución a este obstáculo fue la de emplear la metodología $dividir\ y\ conquistar$ para así poder efectuar el análisis sintáctico en dos etapas. Se encontró que era más eficiente hacer primero una verificación dinámica de las características sintácticas independientes de contexto (evitando así que no se almacene ni se analice un código incorrecto), y segundo, realizar un análisis sintáctico estático de las caraterísticas dependientes del contexto, que hacer el análisis sintáctico en un paso, porque entre otras ventajas, el último proceso de la nueva forma es muy veloz porque tiene la precondición de partir de un código que ya ha pasado por una depuración.

4. Para presentar de manera práctica las propiedades ya demostradas de forma matemática se adelantó la creación de un prototipo en el cual se puedan editar los programas visuales, se efectúen las comprobaciones de las características independientes de contexto, se genere código y se le analicen las características dependientes de contexto al preorden bien fundado que representa el cálculo GraPiCO. Para lo anterior se crearon dos programas: un editor de programas visuales que comprueba el código de manera dinámica, consiguiendo que todos los programas almacenados sean correctos con respecto a las características independientes de contexto y que logra la traducción de un subconjunto de GraPiCO hacia GraPiCO_Textual como un resultado preliminar; y un programa que recibe cualquier programa GraPiCO_Textual y lo compila hacia cálculo PiCO, realizando las comprobaciones sintácticas dependientes de contexto.

En resumen, se puede concluir que se diseñaron de manera formal el cálculo visual GraPiCO y una nueva forma de especificación gramatical (dos puntos donde se ampliaron las fronteras del conocimiento) asegurando que es expresivamente equivalente al cálculo textual PiCO (mediante una Correspondencia Uno a Uno) y se adelantó la implementación (con

demostración de corrección de sus partes garantizando un comportamiento y resultado adecuado) de un prototipo de GraPiCO donde se disminuyó la complejidad del proceso de compilación de sus predecesores mediante el mecanismo de $dividir$ y $conquistar$ que queda contemplado en el diseño de dos nuevos procesos eficientes de análisis sintáctico y en la introducción de sus correspondientes reglas de traducción.

8.3. Trabajo futuro

La meta final de este trabajo de investigación es adicionar todas las funcionalidades al cálculo GraPiCO para convertirlo en un lenguaje de programación; estas características se listan a continuación:

- Programación de las traducciones de los constructores visuales restantes en el editor de GraPiCO. Este es el primer trabajo que se hará porque es primario posibilitar la generación de un programa completo en términos de GraPiCO_Textual para los estudios posteriores (en este momento ya se hace la dirección de un grupo de desarrollo que está haciendo dicha labor).

- Dado que el código objeto actual es el cálculo PiCO (lenguaje no ejecutable directamente) se requiere, una etapa de transformación de PiCO hacia código de la nueva máquina virtual presentada en [SL07] para su ejecución. Este trabajo puede representar una implementación relativamente simple si se tiene en la cuenta que existe una gramática LL(1) del cálculo PiCO y esto garantiza que es posible la realización de traducción dirigida por la sintasis mediante análisis descendente predictivo sin recursión.

- Implementación de una etapa de optimización de código que aproveche las características formales del cálculo PiCO y la estructura de la máquina virtual implementada en [SL07]. La parte difícil es que esta tarea requiere un estudio sobre reglas de optimización de lenguajes concurrentes, pues existen investigaciones al respecto pero los resultados no han sido difundidos en ambientes académicos sino que están disponibles en el comercio. La parte que sería más simple es la de unir los resultados de las investigaciones sobre optimización de lenguajes orientados al objeto y por restricciones.

- Implementación de una interfaz de ejecución que permita observar de manera dinámica la computación de un programa GraPiCO que siga su sistema de transición (reglas de reducción). La idea es poder ejecutar un programa como una animación al estilo de Pictorial Janus; este cometido es el computacionalmente más exigente porque aunque en este documento se presentan las reglas de reducción del cálculo GraPiCO, no es un trabajo simple dibujar la transformación de los constructores asociados.

- Adicionar constructores para aumentar su expresividad o para cumplir con los requisitos de algún paradigma de programación. Algunos ejemplos serían: clases, datos como cadenas y conjuntos, y sentencias de control. Es importante brindar la posibilidad de emplear constructores de paradigmas como los orientados a objeto, funcional, e incluso lógico para lograr aprovechar los desarrollos en los otros lenguajes de programación y la experiencia adquirida por sus usuarios.

Bibliografía

[Neb01] J. Nebrijo. Cortex. In *Trabajo de grado, UNIVALLE, Facultad de Ingeniería, Escuela de Ingeniería de Sistemas y Computación*, Cali-Valle, Colombia, 2001.

[ABH98] G. Álvarez, A. Buss, and M. Heredia. MAPiCO: An abstract machine for the PiCO calculus. In *Cuarto informe de avance, Modelo para la integración de paradigmas orientado_objetos y satisfacción de restricciones en un lenguaje visual. Colciencias*, Cali-Valle, Colombia, 1998.

[ADQ+98] G. Álvarez, J. F. Díaz, L. Quesada, F. Valencia, C. Rueda, and G. Assayag. PiCO: A calculus of concurrent constraint objects for musical applications. In *Workshop on Constraints and the Arts, ECAI98*, Brighton, England, 1998.

[ADQ+01] 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. In *Constraints Journal, Vol 6, No. 1, (2001), pp.21-52.*, Amsterdam, 2001.

[AQRT98] G. Álvarez, L. Quesada, C. Rueda, and G. Tamura. A visual calculus for the cordial language. In *Cuarto informe de avance, Modelo para la integración de paradigmas orientado_objetos y satisfacción de restricciones en un lenguaje visual. Colciencias*, Cali-Valle, Colombia, 1998.

[BAD+01] M. Burnett, J. Atwood, R. Djang, H. Gottfried, J. Reichwein, and S. Yang. Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. In *Journal of Functional Programming*, 2001.

[BFB90] A. Borning and B. Freeman-Benson. Kaleidoscope: Mixin objects, constrains and imperative programing. In *Proceedings of the European conference on object-oriented programming on object-oriented programming systems, languajes, and applications, ottawa, Canada*, 1990.

[Bol01] H. Boley. The rule markup languaje: Rdf-xml data model, xml schema hierarchy, and xsl transformations. In *International Conference on Applications of Prolog - INAP, Tokyo*, October 2001.

[Bor81] A. Borning. The programing language aspects of thinglab. In *ACM Transactions on Programing Languages and Systems*, 1981.

[BS16] C. Bally and A. Séchehaye. Cours de linguistique générale. 1916.

[Cat99] N. Cata no. El compilador PiCO-MAPiCO. In *Trabajo de grado, UNIVALLE, Facultad de Ingeniería, Escuela de Ingeniería de Sistemas y Computación*, Cali-Valle, Colombia, 1999.

[CC90] G. Costagliola and S. K. Chang. Dr parsers: A generalization of lr parsers. In *Proceedings of the 1990 IEEE Workshop on Visual Languages. IEEE Comp. Soc. Press,*, 1990.

[CDFG01] G. Costagliola, V. Deufemia, F. Ferrucci, and C. Gravino. On the plr parsability of visual languages. In *IEEE Symposia on Human-Centric Computing, Stresa, Italy, September*, 2001.

[CGs+91] C. Crimi, A. Guercio, G. Nota, G. Pacini, G Tortora, and M. Tucci. Relational grammars and their application to multidimensional languages. In *Journal of Visual Languages and Computing, 2:333-346.*, 1991.

[Cha88] S. K. Chang. The design of a visual language compiler. In *Proceedings of 1988 IEEE Workshop on Visual Languages, Pittsburgh, October, 84-91*, 1988.

[Cha90] S. K. Chang. Principles of visual programming systems. In *Prentice-Hall*, 1990.

[Cha99] S. K. Chang. Using linear positional grammars for the lr parsing of 2-d symbolic languages. In *appear in GRAMMAR*, 1999.

[CLO94] G. Costagliola, A. De Lucia, and S. Orefice. Towards efficient parsing of diagrammatic languages. In *Proceedings of the 1994 International Workshop on Advanced Visual Interfaces, pages 162-171, ACM Press.*, 1994.

[Col90] A. Colmerauer. An introduction to prolog iii. In *Communications of the ACM, Volume 33, Issue 7, July*, 1990.

[Cos00] G. Costagliola. Extended positional grammars. In *Proceedings of the IEEE International Symposium on Visual Languages*, 2000.

[CP99] Y. Colmenares and G. Puerta. Analizador sintáctico del lenguaje cordial utilizando el formalismo gramatical posicional plr. In *Tesis de pregrado, U. JAVERIANA, Facultad de Ingeniería, Programa Ingeniería de Sistemas y Computación*, Cali-Valle, Colombia, 1999.

[CP01] G. Costagliola and G. Polese. Extended positional grammars. In *Tecnical report, Dipatimento di Matematica ed Informatica, University of Salerno, Salerno, Italy*, 2001.

[CTOL97] G. Costagliola, G. Tortora, S. Orefice, and A. De Lucia. A parsing methodology for the implementation of visual systems. In *Tecnical report, Dipatimento di Informatica ed Applicazioni, University of Salerno, Salerno, Italy.*, 1997.

[Din88] M. Dincbas. The constrain logic programing language chip. In *Tecnical report, TR-LP-37, ECRC, Munich, Germany, May*, 1988.

[DYP92] J. Darlington, Y.Guo, and H. Pull. A new perspective on integrating functional and logic languages. In *Fifth Gerenation Computer System, pages 682-693*, Tokyo,Japan, 1992.

[EK05] M. Erwig and S. Kollmansberger. Visual specifications of correct spreadsheets. In *IEEE Symp. on Visual Languages and Human-Centric Computing, Dallas, TX, USA*, September 2005.

[Erw97] M. Erwig. Semantics of visual languages. In *13 th IEEE Symp. on Visual Languages, Capri*, 1997.

[Erw98a] M. Erwig. Abstract syntax and semantics of visual languages. In *Journal of Visual Languages and Computing, Vol 9, No. 5, 461-483*, 1998.

[Erw98b] M. Erwig. Abstract syntax and semantics of visual languages. In *Journal of Visual Languages and Computing*, 1998.

[Erw98c] M. Erwig. Visual semantics - or: What you see is what you compute. In *14 th IEEE Symp. on Visual Languages*, 1998.

[Erw06] M. Erwig. Visual type inference. In *Journal of Visual Languages and Computing*, 2006.

[FNFS06] A. Fernández, A. Navarro, B. Fernández, and J.L. Sierra. Lenguajes de programación, lenguajes de marcado y modelos hipermedia: Una visión interesada de la evolución de los lenguajes informáticos. In *Universidad Complutense, Madrid*, 2006.

[Gol91] E. J. Golin. Parsing visual languages with picture layout grammars. In *Journal of Visual Languages and Computing, 2:1-23.*, 1991.

[GT98] J. Gutiérrez and C. Tavera. Primera versión del sistema de restricciones para la máquina abstracta MAPiCO, lógica de primer orden, método de las bolsas. In *Cuarto informe de avance, Modelo para la integración de paradigmas orientado_objetos y satisfacción de restricciones en un lenguaje visual. Colciencias*, Cali-Valle, Colombia, 1998.

[GV01] C. García and A. Vásquez. Implementación eficiente de un sistema de restricciones de dominios finitos para el lenguaje cordial. In *Trabajo de grado, U. JAVERIANA, Facultad de Ingeniería, Plan Ingeniería de Sistemas*, Cali-Valle, Colombia, 2001.

[Hir06] D. Z. Hirtle. Translator: a translator from language to rules. In *Canadian Symposium on Text Analysis -CaSTA, Fredericton*, October 2006.

[JL87] J. Jaffar and J. Lassez. Constrains logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programing Languages, pages 111-119*, Munich, Germany, 1987.

[JP02] C. Jiménez and D. Polo. Proceso de reingeniería en el editor de cordial. In *Tesis de pregrado, U. JAVERIANA, Facultad de Ingeniería, Programa Ingeniería de Sistemas y Computación*, Cali-Valle, Colombia, 2002.

[KKR95] P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. In *Journal of Computer and Systems Science*, 1995.

[Mar94] K. Marriot. Constraints multiset grammars. In *Proceedings of the 1995 IEEE Symposium on Visual Languages, pages 118-125.IEEE Comp. Soc. Press*, 1994.

[Mil93] R. Milner. The polyadic $\pi-$ calculus: A tutorial. In *Technical Report ECS-LFCS-91-180, Laboratory for Foundations of Computer Science. Also in Logic and Algebra of Specification, F. L. Bauer, W. Brauer and H. Schwichtenberg, editors, Springer Verlag*, 1993.

[MPPJ05] A. Martinez, F. Perez, M. Patino, and R. Jimenez. A visual web service composition tool for bpel4ws. In *IEEE Symposium on Visual Languages and Human-Centric Computing, Dallas, TX, USA*, September 2005.

[Nad04] M. Nadin. Technical report, texas university at dallas. In *Visual Programming Languages - Efficiency of the Visual driving Technology*, Dallas, U.S.A, 2004.

[owl04] Owl web ontology language guide. In *first ed. W3C Recommendation*, 10 febrary 2004.

[PA03] C. Pautasso and G. Alonso. Visual composition of web services. In *IEEE Symposia on Human-Centric Computing Languages and Enviroments, Auckland, New Zealand*, October 2003.

[PA05] C. Pautasso and G. Alonso. Jopera: an agile environment for web service composition with visual unit testing and refactoring. In *Journal of Visual Languages and Computing*, 2005.

[Phi07] C. Philip. Enhancing the programmability of spreadsheets with logic programming. In *IEEE Symposium on Visual Languages and Human Centric Computing, Coeur dÁlne*, September 2007.

[PN08] C. Philip and P. Nicholson. Unification of arrays in spreadsheets with logic programming. In *Symposium on the Practical Aspects of Declarative Languages, San Francisco, California, USA*, January 2008.

[QR98] L. Quesada and C. Rueda. A denotational semantics of PiCO. In *Cuarto informe de avance, Modelo para la integración de paradigmas orientado_objetos y satisfacción de restricciones en un lenguaje visual. Colciencias*, Cali-Valle, Colombia, 1998.

[Rag05] D. Raggett. Getting started with html. In *World Wide Web Consortion*, May 24 2005.

[Sar93] V. Saraswat. Concurrent constraint programming. In *ACM Distinguished Dissertation Series*, 1993.

[SL07] A. Sarasti and C. Llano. Reimplementación de la máquina abstracta MAPiCO. In *Tesis de pregrado, U. JAVERIANA, Facultad de Ingeniería, Programa Ingeniería de Sistemas y Computación*, Cali-Valle, Colombia, 2007.

[Smo95] G. Smolka. The oz programming model. In *Lecture Notes in Computer Science, vol. 1000, pages 324-343.*, Springer-Verlag,Berlin, 1995.

[Ste80] G. Steele. The definition and implementation of a computer programing language based on constraints. In *Phd Thesis, Dept. of Electrical Engienerring and Computer Science, M.I.T.*, 1980.

[TD06] C. Tavera and J. Díaz. Alternativa para especificación sintáctica: Gramática de sistemas de íconos generalizados: Gsig. In *Congreso Argentino de Ciencias de la Computación, San Luis, Argentina*, Octubre de 2006.

[TD09] C. Tavera and J. Díaz. Alternativa de mecanismo de traducción de lenguajes mediante análisis de sámbolos de sincronizacián: T_Gsig. In *Revista Ingeniería y Desarrollo, Universidad del Norte, Barranquilla, Colombia, Número 26*, Diciembre de 2009.

[TD07] C. Tavera and J. Díaz. Nuevo cálculo visual: Grapico. In *Congreso Colombiano de Computación, Bogotá, Colombia*, Abril de 2007.

[TD08] C. Tavera and J. Díaz. Breve discusión de las ventajas de los lenguajes visuales frente a los textuales: Caso de estudio el cálculo grapico. In *Congreso Colombiano de Computación, Medellin, Colombia*, Abril de 2008.

[TDS+07] C. Tavera, J. Díaz, A. Soto, J. Gallego, and A. Jojoa. Alternativa de comprovación sintáctica de vlps: Gsig_parsing. In *Congreso Argentino de Ciencias de la Computación, Corrientes y Resistencia, Argentina*, Octubre de 2007.

[Tea98] AVISPA Research Team. The implementation of the cordial editor, progress report. In *Cuarto informe de avance, Modelo para la integración de paradigmas orientado_objetos y satisfacción de restricciones en un lenguaje visual. Colciencias*, Cali-Valle, Colombia, 1998.

[TVG94] M. Tucci, G. Vitiello, and G.Costagliola. Parsing nonlinear languages. In *IEEE Transactions on Software Engineering, 20(9):720-739.*, 1994.

[Wit92] K. Wittenburg. Earley style parsing for relational grammars. In *Proceedings of the 1992 IEEE Workshop on Visual Languages, pages 192-199.IEEE Comp. Soc. Press,*, 1992.

[Wit96] K. Wittenburg. Relational grammars: Theory and practice in a visual language interface for process modeling. In *AVI 96 International Workshop on Theory of Visual Languages, May 30, Gubio, Italy*, 1996.

[XML06] eXtensible Markup Language (xml) 1.0. In *fourth ed. W3C Recommendation*, 29 september 2006.

Los humanos al enfrentar un problema primero imaginan la solución y luego, a partir de estos dibujos creados con el pensamiento, construyen los resultados: cocinan alimentos, diseñan vehículos, escriben poemas y componen música. Lo visual es inherente en las personas, por esta razón las primeras manifestaciones realmente escritas de la humanidad se presentaron 14 siglos después que las gráficas y que l@s niñ@s hacen dibujos tres años antes de sus textos iniciales.

En el terreno tecnológico la evolución se dio de manera inversa; primero se procesaron textos y luego imágenes. Las primeras impresoras sólo podían plasmar unas rudimentarias letras y ni pensar en dibujos; los monitores en sus orígenes únicamente presentaban pocas líneas de texto; los lenguajes de programación y los sistemas operativos iniciales ni siquiera tenían entorno gráfico y los electrodomésticos no llevaban pantallas que desplegaran información; esto sucedía porque aunque lo gráfico es innato en los seres humanos, para manipular artificialmente figuras se requieren niveles de procesamiento muy superiores. Este último es el motivo por el cual, aunque las teorías de la geometría computacional y de procesamiento de imágenes ya estaban muy avanzadas en el siglo XX, apenas ahora los procesadores son lo suficiente veloces y económicos como para poder ejecutar los sistemas operativos totalmente visuales y los lenguajes de programación realmente gráficos en sus albores.

ISBN: 978-958-8436-35-7
9 789588 436357

Universidad de San Buenaventura,
seccional Cali.
La Umbría, carretera a Pance
PBX: 318 22 00 - 448 22 22. Fax: 555 20 06
A.A. 7154 y 25162.
www.usbcali.edu.co