Herramienta visual para la enseñanza de programación estructurada

Champredonde Raúl[1]
De Giusti Armando[2]
Laboratorio de Investigación y Desarrollo en Informática[3]
Departamento de Informática
Facultad de Ciencias Exactas
Universidad Nacional de La Plata

1. Resumen

Se presenta un lenguaje de programación junto con su ambiente de desarrollo visual. El objetivo del lenguaje es el aprendizaje de la programación estructurada a nivel introductorio.

En la especificación e implementación del mismo se ha puesto énfasis en la asimilación, por parte del estudiante, de aquellos aspectos de la programación que hacen a un buen estilo y que favorecen la legibilidad, el mantenimiento, la adaptabilidad.

El ambiente de desarrollo permite la implementación y ejecución de algoritmos especificados en el lenguaje mencionado.

La implementación de un algoritmo puede llevarse a cabo en forma convencional, es decir, escribiendo cada una de las instrucciones que lo componen, o por medio de la confección de un diagrama visual, el cual a su vez, sirve como herramienta de diseño.

Ambas formas de implementación pueden utilizarse indistintamente en cualquier momento del desarrollo de un programa, manteniéndose tanto el diagrama visual como el código, permanentemente actualizados. Esto significa que toda modificación en uno cualquiera de los dos se refleja en forma inmediata y automática en el otro.

[1] Jefe de Trabajos Prácticos, Dedicación Semiexclusiva, LIDI, Dep. de Informática, Fac. de Cs. Exactas, UNLP. E-mail: rchampre@ada.info.unlp.edu.ar
[2] Prof. Titular Dedicación Exclusiva. Investigador Principal del CONICET. Director del LIDI, Dep. de Informática, Fac. de Cs. Exactas, UNLP. E-mail: degiusti@ada.info.unlp.edu.ar
[3] LIDI, Laboratorio de Investigación y Desarrollo en Informática, Dep. de Informática, Fac. de Cs. Exactas, UNLP. E-mail: lidi@ada.info.unlp.edu.ar

2. Introducción

El objetivo de este trabajo está centrado en la enseñanza de los primeros conceptos de programación impartidos a lo largo del curso de ingreso de las carreras Licenciatura en Informática y Analista de Computación de la Facultad de Ciencias Exactas de la Universidad Nacional de La Plata y en la cátedra Programación de Computadoras de las mismas carreras.

Desde hace varios años, se introducen los conceptos principales de la programación estructurada utilizando los resultados de una línea de trabajo que se detalla en una serie de artículos publicados en congresos del país y del exterior, especificaciones, informes técnicos, apuntes todos ellos disponibles en el LIDI.

Esos resultados son suficientes para la enseñanza de la programación estructurada pero se considera necesario responder a la evolución tecnológica en cuanto a herramientas de desarrollo y paradigmas de programación se refiere.

El mundo de las herramientas de desarrollo ha sufrido un positivo vuelco hacia los ambientes visuales. Basta con ver la cantidad y variedad de productos existentes en el mercado que hacen uso de esta técnica, para reconocer su éxito.

Por otro lado, en la última década ha tomado gran vigor la orientación a objetos y la programación concurrente.

Estas tendencias, actualmente bien consolidadas, no pueden dejar de ser consideradas como parte de la educación que reciben los alumnos del primer año de una carrera universitaria de informática.

Este trabajo, que junto con otros [MADO, 1996] forma parte de la continuación de la línea de trabajo mencionada, implementa un ambiente visual que permite al alumno desarrollar algoritmos y probarlos, fomentando un buen estilo de programación, evitando la adquisición de vicios y alentando la utilización de técnicas y herramientas de diseño e implementación.

Además se intenta que el alumno se familiarice paulatinamente con el uso de herramientas visuales de desarrollo [KOLD, 1994] [CHAM, 1995] y con ciertas técnicas de análisis y diseño cuya utilización está creciendo considerablemente a nivel mundial.

En la especificación e implementación de este ambiente se debió tener en cuenta su futura extensión para encarar la enseñanza de temas relacionados con la programación en general, con distintos paradigmas de programación, con la programación concurrente y con la utilización de técnicas visuales de análisis y diseño.

Además, y por sobre todo, se debió considerar la posibilidad de manipular un robot real con capacidades de translación, rotación y percepción (como mínimo hacia adelante). Aunque no es indispensable, sería deseable contar aún con otro robot: un brazo de al menos dos grados de libertad montado sobre el anterior. Con estos elementos los resultados de la ejecución de un algoritmo, además de verse en la pantalla de una computadora, sería vista también en forma real sobre la maqueta de una ciudad.

3. Antecedentes

Durante varios años investigadores del Laboratorio de Investigación y Desarrollo en Informática (LIDI) han trabajado en la especificación de un robot abstracto que se lo denominó Lubo-1 y un lenguaje de programación estructurada para controlar el accionar de dicho robot.

El robot puede caminar por las calles o avenidas de una ciudad (Figura 1) describiendo distintas trayectorias dependiendo del algoritmo que lo controla. Dicha ciudad tiene cien avenidas y cien calles, siendo las primeras verticales y horizontales las segundas. Cada paso del robot lleva a este de una esquina a otra según su dirección. Una esquina es la intersección de una calle con una avenida.

En su camino el robot puede recoger y depositar papeles y flores en las esquinas de la ciudad. Para ello cuenta con dos bolsas, una de flores y otra de papeles, en las que pone, respectivamente los elementos que recoge de las esquinas, y de las que toma las flores y los papeles que deposita. Obviamente para poder depositar una flor o un papel deberá tener en cuenta previamente que la bolsa respectiva no se encuentre vacía, pues de lo contrario puede ocurrir un error en ejecución.

Figura 1

Además, en las esquinas de la ciudad puede haber obstáculos que no permiten el paso del robot. Por lo tanto debe, necesariamente, verificar la existencia o inexistencia de un obstáculo en la esquina hacia la cual desea avanzar antes de hacerlo. Si se intenta pasar por una esquina en la que se encuentra un obstáculo, se produce un error en ejecución.

Los obstáculos pueden aparecer en esquinas consecutivas formando lo que se denomina una barrera. El robot también debe tenerlas en cuenta de manera tal que no "choque" con una de ellas en su recorrido.

La sintaxis del lenguaje se encuentra brevemente descripta en el anexo al final del artículo.

Inicialmente, se tienen en cuenta sólo dos tipos de datos: valores numéricos y valores lógicos. Esto permite proponer una gran variedad de problemas pero sin desviar la atención del estudiante a cuestiones ajenas al algorítmo en sí y a su estructura, pero permitiendo la representación de expresiones de diversas complejidades.

A medida que se avance en esta línea de investigación se integrarán otros tipos de datos primitivos, tipos de datos definidos por el usuario, tipos abstractos y clases y objetos de manera tal que este trabajo evolucione siguiendo los conceptos que se imparten en los primeros años de la carrera.

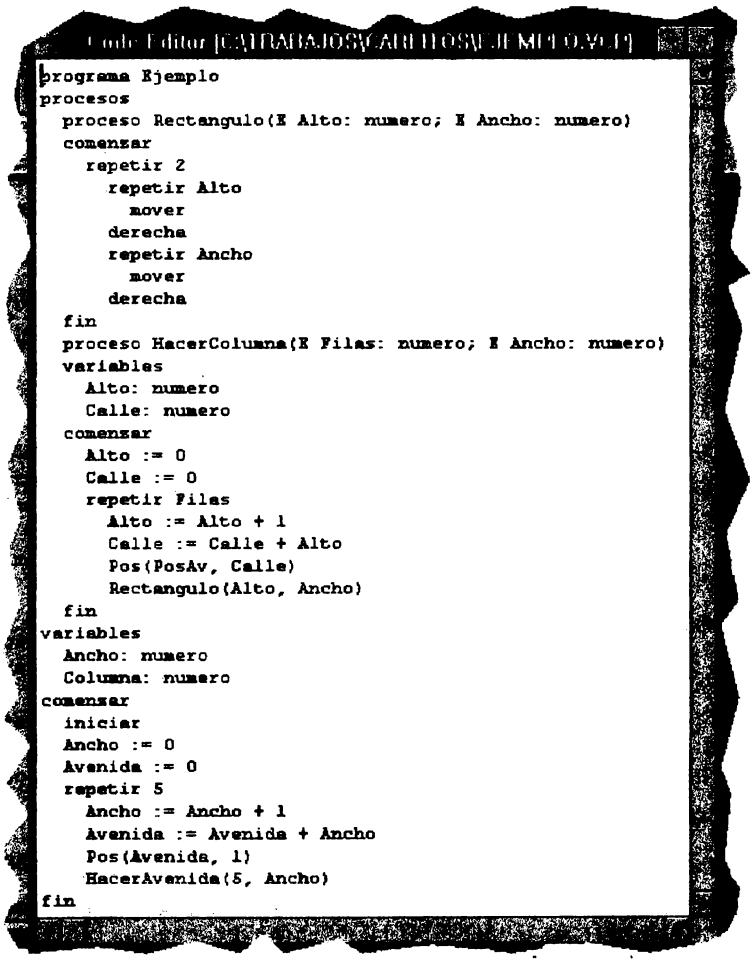
No se utilizan palabras claves para agrupar un conjunto de instrucciones simples en una única sentencia compuesta (como *begin / end* de Pascal, *{ / }* de C), sino que se determina según la indentación de cada instrucción, siguiendo, en este sentido, reglas sintácticas similares a la de Occam [OLSE, 1983] [CSA, 1990]. Esto exceptúa al programa

principal y a los subprogramas cuyos cuerpos quedan delimitados por las palabras claves *comenzar / fin.*

Nuestra experiencia docente indica que, en la mayoría de los casos, los estudiantes no ponen mayor atención en detalles como la indentación, lo cual redunda en algoritmos desordenados y por tanto contradictorios con los principios de legibilidad, mantenimiento, adaptabilidad, etc.

Definiendo el lenguaje con esta forma de agrupamiento, se garantiza que el usuario debe necesariamente poner atención en la indentación hasta que, con el tiempo, se acostumbra práctica y visualmente a desarrollar algoritmos con cierto orden y estilo.

A manera de ejemplo, la figura 2 muestra un algoritmo implementado en este lenguaje.



```
programa Ejemplo
procesos
  proceso Rectangulo(E Alto: numero; E Ancho: numero)
  comenzar
    repetir 2
      repetir Alto
        mover
      derecha
      repetir Ancho
        mover
      derecha
  fin
  proceso HacerColumna(E Filas: numero; E Ancho: numero)
  variables
    Alto: numero
    Calle: numero
  comenzar
    Alto := 0
    Calle := 0
    repetir Filas
      Alto := Alto + 1
      Calle := Calle + Alto
      Pos(PosAv, Calle)
      Rectangulo(Alto, Ancho)
  fin
variables
  Ancho: numero
  Columna: numero
comenzar
  iniciar
  Ancho := 0
  Avenida := 0
  repetir 5
    Ancho := Ancho + 1
    Avenida := Avenida + Ancho
    Pos(Avenida, 1)
    HacerAvenida(5, Ancho)
fin
```

Figura 2

Muchas veces se enseña antes el concepto de variables globales que el de pasaje de parámetros. Esto puede resultar perjudicial para la formación de los alumnos. Además de los posibles errores colaterales que el uso de variables globales implica, el alumno se siente cómodo con su uso y, llegado el caso de tener que comunicar distintos módulos de un programa, la primer solución en la que piensa es variables globales.

Revertir esta situación es relativamente costoso y más aún si se piensa que el tiempo que ello lleva podría ser utilizado para avanzar en otros conceptos.

Por eso es conveniente no permitir el uso de variables globales, al menos hasta que estén totalmente asimilados los distintos mecanismos de pasaje de parámetros y que el uso de los mismos surja en forma natural. Luego habrá tiempo para mostrar a través de ejemplos cómo reemplazar parámetros por variables globales y cuáles son las desventajas que esto implica.

Como una forma de reforzar esta idea, los procedimientos, llamados procesos en el lenguaje, deben ser definidos antes que las variables. De esta manera, las variables del programa actúan como locales al mismo, no pudiendo ser referenciadas desde el cuerpo de los procedimientos.

Los parámetros formales de un proceso están definidos por su nombre, tipo y modo. El modo puede ser *Entrada*, *Salida* o *EntradaSalida*, al estilo del pasaje de parámetros de Ada [OLSE, 1983] [USD, 1983]. Con estos tres modos se cubren las posibilidades más comunes de pasaje de parámetros de los lenguajes de programación.

4. Objetivo

El objetivo final de esta línea de investigación es lograr un producto de los denominados "Computer Based Training" para los primeros pasos de la carrera de informática.

Por eso se ha desarrollado un sistema multimedia interactivo con todos los conceptos que se desean enseñar y el ambiente de desarrollo que aquí se describe, complementándose para que en un mismo producto se encuentren los conceptos necesarios, refuerzos de aquellos no totalmente comprendidos, ejemplos y la posibilidad de probar y experimentar.

Asimismo, este trabajo sienta las bases para las extensiones previstas, entre las cuales se pueden mencionar la incorporación de un robot real y la posibilidad de contar con varios robots en una misma ciudad para la enseñanza de programación concurrente.

5. Descripción del ambiente

El ambiente corre bajo Windows utilizando las mismas características de la interfaz que la mayoría de las aplicaciones que corren bajo Windows. Esto contribuye a que el alumno se habitúe a un ambiente gráfico similar al que deberá utilizar en su vida profesional, pero por el momento, sin ninguna necesidad de interiorizarse en los detalles específicos de este tipo de entornos o sistemas operativos.

El aspecto general del ambiente de desarrollo (Figura 3) es semejante al de los productos que se autodenominan "visuales". En particular sigue las características principales del Borland Delphi [CALV, 1995] [LUDO, 1995].

Está integrado por una ventana principal con su menú y barra de botones generales, una ventana de edición del código, otra de edición del diagrama visual, y por último una ventana con la ciudad por la que transita el robot, en la que se puede ver el recorrido del mismo durante la ejecución de un algoritmo.

Se provee el manejo de archivos, impresión, posibilidades de edición y búsqueda, selección de distintas opciones de visualización y configuración del ambiente, ayuda en línea, etc., habituales en la mayoría de las herramientas de desarrollo.

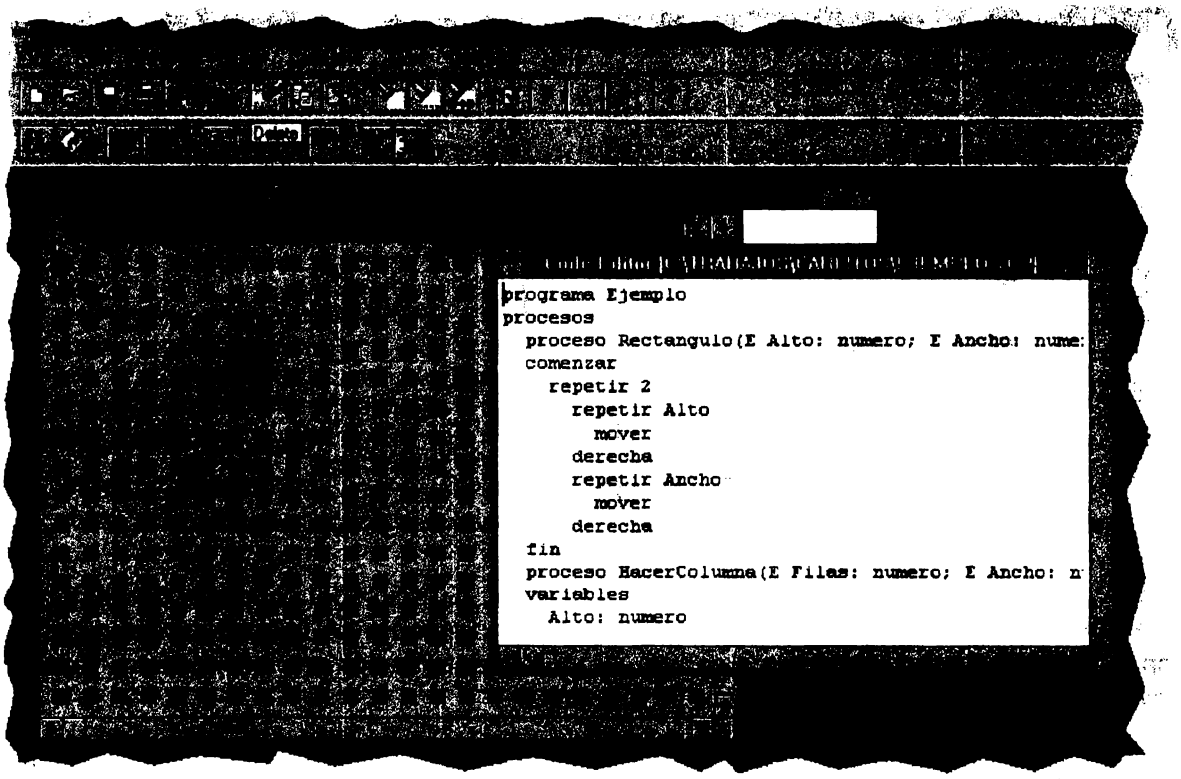


Figura 3

Naturalmente se provee también comprobación de sintaxis, ejecución paso a paso y ejecución completa. El resultado de la ejecución en cualquiera de sus formas puede ser apreciado en la ventana que contiene a la ciudad, mientras que en las ventanas de edición del código y de edición del diagrama visual se denota la instrucción del algoritmo que está siendo ejecutada corrientemente.

6. Diagrama Visual Estructurado

Como fue expuesto anteriormente, es posible desarrollar un algoritmo escribiendo cada una de las sentencias que lo componen en el editor de código, o bien mediante la confección visual de un diagrama que se crea y modifica en el editor de diagramas. El código y el diagrama se encuentran permanentemente sincronizados, por lo cual es posible pasar de uno a otro en cualquier punto de un desarrollo, sin necesidad de preocuparse por las últimas modificaciones.

Para la confección del diagrama visual se hace uso de los elementos disponibles en la segunda barra de botones (Figura 3). Hay un botón para cada instrucción, estructura de control y constructores necesarios para la elaboración de un algoritmo.

Las figuras 4a y 4b muestran el diagrama visual asociado a uno de los subprogramas especificados en el código de la figura 2.

El desarrollo visual de un algoritmo se realiza por medio de la inserción de íconos en el diagrama, seleccionando un elemento y depositándolo en la sección deseada del diagrama.

Un programa tiene una parte de declaraciones de subprogramas, otra de declaraciones de variables y una tercera para la especificación del cuerpo.

En la parte de declaraciones de subprogramas sólo se los especifica. Para la implementación de cada uno de ellos se utiliza el editor de diagramas como si se tratara de

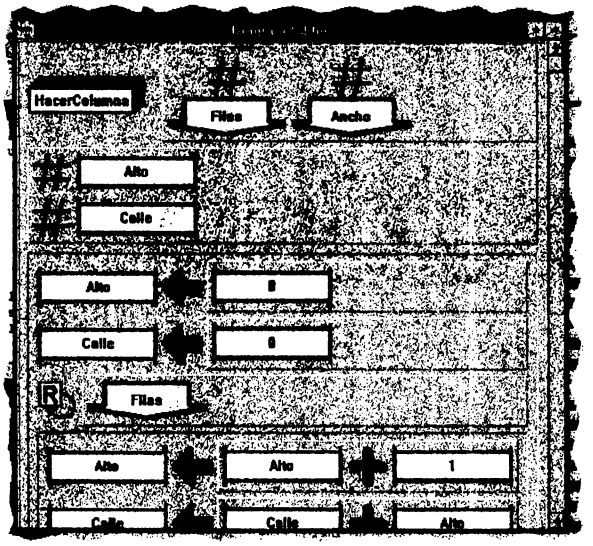


Figura 4a

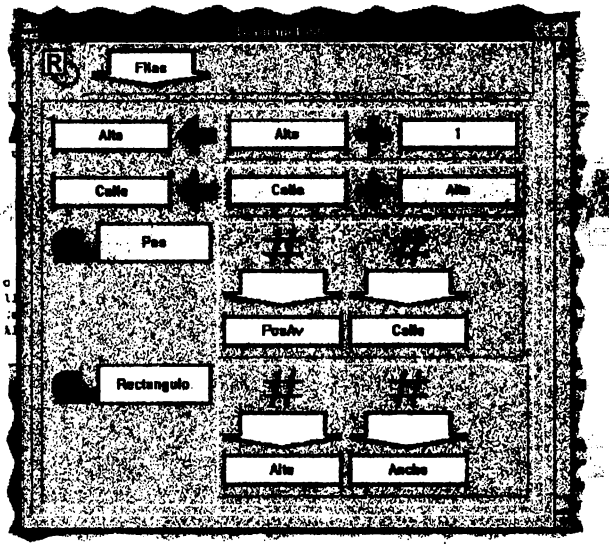


Figura 4b

un programa completo. El fundamento de esto es la utilización del concepto de abstracción. Cuando se implementa visualmente un programa solamente se utilizan los subprogramas especificados sin ocuparse en absoluto de la implementación de los mismos. Esta idea también vale para la implementación de procesos.

Los subprogramas tienen las mismas secciones que los programas pero además tienen otra destinada a la especificación de los parámetros formales.

Cada una de las secciones mencionadas está representada por un panel tridimensional. La tridimencionalidad es utilizada como una forma de reforzar la idea de profundidad de las instrucciones y para resaltar las definiciones que deberán ser tenidas en cuenta durante el desarrollo. Las especificaciones de parámetros, variables, subprogramas y expresiones se encuentran en paneles sobresalientes, mientras que los cuerpos de los programas, subprogramas y estructuras de control, en paneles de mayor profundidad.

Cada estructura de control tiene asociada una expresión aritmética o lógica según corresponda. Esta última, está representada a un lado del ícono de la estructura de control, dentro de un panel tridimensional.

La inserción de una primitiva simplemente incorpora el ícono correspondiente.

Las invocaciones a subprogramas tienen a su lado un panel que contiene una indicación del modo y tipo de cada uno de los parámetros formales, a los cuales se asocian los parámetros actuales.

La utilización del diagrama visual estructurado flexibiliza sumamente el desarrollo de un programa, permitiendo la utilización de distintas metodologías para la definición de un algoritmo.

En un extremo, está la posibilidad de definir la estructura general del programa en forma visual utilizando una técnica del tipo "Top-Down" y luego, por medio de refinamientos sucesivos, definir los distintos componentes de esa estructura general, siempre en forma visual. Si bien, de esta manera, no es necesario un contacto directo con el código, el concepto de la estructuración del algoritmo no varía en absoluto.

El extremo opuesto es definir la estructura general del programa y luego refinarla hasta llegar al nivel de las instrucciones por medio de la edición de código.

Lógicamente, todas las alternativas intermedias entre las dos anteriores son variantes válidas para el desarrollo estructurado de programas.

Más allá del nivel de detalle al que se llegue en la definición de un algoritmo en forma visual, la mayor importancia del diagrama visual estructurado está dada por su utilización como herramienta de diseño. A través del diagrama se genera el diseño de algoritmos en forma clara y estructurada, facilitando el mantenimiento y la legibilidad tanto del diseño como del programa en sí.

Es por ello que, simultáneamente con la enseñanza de la programación estructurada, se alienta al estudiante en el uso de herramientas de apoyo al desarrollo de programas, aún en la resolución de problemas sencillos.

Una característica de considerable valor que tiene esta forma de diagrama visual, es la independencia respecto del lenguaje asociado, lo cual claramente permite modificar su sintaxis.

Por ejemplo, como parte del curso de ingreso a nuestra carrera, antes de dar la definición del lenguaje que luego se utilizará, se propone a los alumnos que ellos mismos definan el lenguaje que necesitan para la manipulación de un robot abstracto con las características descriptas.

Con este criterio de independencia del lenguaje, cada uno podría programar el comportamiento del robot con sus propias construcciones sintácticas.

7. Aspectos de Implementación

La implementación de este ambiente fue llevada a cabo utilizando Delphi.

Se eligió el Delphi como herramienta de desarrollo porque permite crear la interfaz de las aplicaciones con gran facilidad, el lenguaje de programación es orientado a objetos (Object Pascal), es suficientemente eficiente en cuanto a tiempos de compilación y también lo es el ejecutable que genera, y tiene las mismas capacidades multimediales y de manejo de bases de datos que otras herramientas similares.

Tales características son, hoy día, muy importantes para el desarrollo de sistemas amigables y flexibles.

Debió analizarse cuidadosamente el tipo de interfaz ya que el usuario final sería un estudiante que, posiblemente estaría en sus primeros contactos con una computadora y, en la mayoría de los casos, su primer experiencia en el uso de un ambiente de desarrollo. Por lo tanto, es un requisito indispensable que el sistema pueda ser fácilmente utilizado en forma intuitiva.

La comprobación de sintaxis así como la ejecución completa o paso a paso, requiere de un tratamiento de cierta complejidad, en el cual intervienen diversas estructuras dinámicas, e interacciones entre varios componentes del sistema. Para ello es muy adecuado el uso de la programación orientada a objetos, sin mencionar las ventajas que ella significa para el desarrollo completo de casi cualquier aplicación.

En la implementación se priorizó la claridad y legibilidad del código por sobre la eficiencia del mismo para facilitar el proceso de adaptación que requieren las extensiones previstas.

De los aspectos de implementación, los más importantes son la comprobación de la sintaxis de los algoritmos y la ejecución de los mismos.

En la comprobación de la sintaxis de un algoritmo se construye una estructura dinámica de objetos. Cada objeto conoce la forma en que debe analizarse. Para ello se definieron objetos tales como TSentencia, TPrimitiva, TAsignacion, TInvocacion,

TSecuencia, TSi, TMientras, TRepetir, TExpresion, TPrograma, TSubprograma, TVariable, TParametro, etc.

Una vez asegurada la corrección del algoritmo, la estructura dinámica de objetos construída durante la comprobación de la sintaxis, es utilizada también para la ejecución del algoritmo. Por eso cada uno de los objetos que representan a los distintos elementos de un algoritmo tienen un método que permite su ejecución.

La utilización de objetos en el desarrollo de este sistema ha aliviado considerablemente su implementación. Cada uno de los objetos de la jerarquía, está dotado de las capacidades necesarias para que él mismo realice su propia comprobación de sintaxis y su ejecución. Si a esto se le suman los beneficios de la herencia y del polimorfismo, se obtiene una considerable reducción de la complejidad del sistema y de la cantidad de líneas de código necesarias.

8. Extensiones y Líneas de Investigación

Una de las extensiones está planeada para la enseñanza de programación concurrente. La idea de base es tener no uno sino varios robots en una misma ciudad, cada uno ejecutando su propio algoritmo, los cuales cooperan para llevar a cabo una determinada tarea y comparten recursos. Por ende, el lenguaje debe proveer mecanismos de sincronización y comunicación entre los robots.

Por ejemplo, uno de los ejercicios más sencillos que se plantean en la práctica es juntar todos los papeles de la ciudad. Con un único robot este problema se solucionaría haciendo que este recorra todas las calles de la ciudad juntando los papeles que encuentre en las esquinas. Si se cuenta con varios robots, se podría utilizar a cien de ellos, cada uno juntando los papeles de una calle.

Para la comunicación y sincronización de los robots que colaboran para un mismo fin, se piensa implementar una variedad de mecanismos: semáforos, regiones críticas, regiones críticas condicionales, monitores, rendezvouz y pasaje de mensajes sincrónico y asincrónico. Este conjunto de mecanismos abarcan una parte importante de la historia de la programación concurrente y de la enseñanza de la misma [ANDR, 1991] [UMAR, 1993].

Otra de la futuras extensiones involucra la incorpración de un robot real al sistema. Se desea utilizar un robot cuyas características pricipales son: 2 motores reversibles de alta resolución (100 ticks por revolución), 7 transductores ultrasónicos de 25 Hz. con sensibilidad de 10 cm. a 5 m., puerto serie RS-232 para entrada/salida externa, 3 entradas digitales, 3 salidas digitales, 1 entrada analógico/digital, 1 timer digital de entrada, 1 timer digital de salida.

Hay dos formas distintas de dar las instrucciones al robot para que este las ejecute. Una de ellas consiste en darle el algoritmo completo y la otra de a una instrucción por vez esperando que termine de ejecutar una para recién después darle la siguiente. En ambos casos cada sentencia debe ser traducida a instrucciones de ejecución directa interpretables por el robot.

En nuestro caso, es necesario utilizar la última forma debido a la indispensable interacción entre el ambiente de desarrollo y el robot. Un ejemplo claro es que la evaluación en tiempo de ejecución de la condición *hay obstaculo* necesita una respuesta por parte del robot, que permita determinar si hay algún objeto frente a uno de sus censores.

Estas extensiones, si bien son las más importantes a los fines didácticos, no son las únicas previstas. También es deseable dotar al lenguaje de otros tipos, tipos definidos por el usuario, tipos abstractos y clases y objetos.

8. Conclusiones

Se ha presentado un ambiente visual para la especificación y ejecución de algoritmos, orientado al aprendizaje de los primeros conceptos de programación.

Se trata de un tema de investigación y desarrollo que se encuentra en plena evaluación.

9. Agradecimientos

Agradecemos a todo el LIDI por su apoyo y en particular al Lic. Hugo Ramón con quien discutimos ciertos aspectos de la implementación del parser.

10. Bibliografía

[ANDR, 1991] Andrews G., Concurrent Programmig. Principles and Practice, The Benjamin/Cummings, 1991.

[CALV, 1995] Calvert C., Delphi Unleashed, Sams Publishing, 1995.

[CSA, 1990] Occam and the Transputer, Computer System Architects, 1990.

[CHAM, 1995] Champredonde R., Koldobsky M., Boracchia M., Lenguajes de Programación Visual y por Demostración, II International Congress on Information Engineering, 1994, Buenos Aires.

[HUNT, 1985] Hunter R., Compilers, Their Design and Construction Using Pascal, John Wiley & Sons, 1985.

[KOLD, 1994] Koldobsky M., Champredonde R., De Giusti A., ADOOVE: Un Ambiente Visual por Demostración Orientado a Objetos de Derivación Automática, 23as. Jornadas de Informática e Investigación Operativa, 1994, Buenos Aires.

[LUDO, 1995] Ludovic D., Delphi-Desarrollo Rápido de Aplicaciones Bajo Windows, Eyrolles, 1995.

[MADO, 1996] Madoz C., Gorga G., Bertone R., De Giusti A., Diseño de Experiencias de Autoaprendizaje con Herramientas Multimediales en el Ingreso a Informática, 1996.

[OLSE, 1983] Olsen E., Whitehill S., Ada for programmers, Prentice-Hall, 1983.

[SETH, 1989] Sethi R., Programming Languages. Concepts and Constructs, Addison-Wesley, 1989.

[UMAR, 1993] Umar A., Distributed Computing and Client-Server Systems, Prentice Hall, 1993.

[USD, 1983] Reference Manual for the Ada Programming Languaje, ANSI-MIL-STD-1815A, U.S. Department of Defense, 1983.

Anexo: sintaxis del lenguaje

La siguiente es una breve descripción de la sintaxis del lenguaje.

El mismo está compuesto por las estructuras de control:

- *si <condición>*

  <secuencia de sentencias>

- *si <condición>*

  < secuencia de sentencias>

  sino

  < secuencia de sentencias>

- *repetir <nro. de veces>*

  < secuencia de sentencias>

- *mientras <condición>*

  < secuencia de sentencias>

las cuales tienen el mismo comportamiento y características que las estructuras de control de los lenguajes procedurales convencionales; las primitivas:

- *Mover*: lleva el robot de la esquina en la que se encuentra a la siguiente en la dirección hacia la cual está orientado. Si se intenta ejecutar esta instrucción cuando Lubo-1 se encuentra en un borde de la ciudad orientado hacia el exterior de sus límites, ocurrirá un error en tiempo de ejecución.

- *Derecha*: orienta al robot hacia la dirección de la derecha con respecto a la orientación corriente. La inexistencia de una instrucción análoga Izquierda fuerza al estudiante a definir su propia forma de hacer girar al robot hacia la izquierda.

- *Depositar flor*: saca una flor de la bolsa de flores y la deposita en la esquina en la que se encuentra. Si se intenta ejecutar esta instrucción cuando la bolsa de flores está vacía, se produce un error en tiempo de ejecución.

- *Depositar papel*: saca un papel de la bolsa de papeles y lo deposita en la esquina en la que se encuentra. Si se intenta ejecutar esta instrucción cuando la bolsa de flores está vacía, se produce un error en tiempo de ejecución.

- *PosAv*: devuelve el valor de la avenida sobre la cual se encuentra el robot.

- *PosCa*: devuelve el valor de la calle sobre la cual se encuentra el robot.

- *Pos(Av, Ca)*: posiciona al robot en la esquina determinada por los parámetros Av y Ca.

las condiciones booleanas especiales:

- *hay flor*
- *hay papel*
- *hay flor en la bolsa*
- *hay papel en la bolsa*

cuya semántica resulta ovbia, además de las operaciones aritméticas y lógicas convensionales, y las expresiones ariméticas y lógicas especificadas en la forma usual, y el contructror de subprogramas:

- *proceso: palabra clave que denota la definición de un subprograma. El mismo deberá* estar seguido del nombre del subprograma y sus parámetros, indicando para cada uno de ellos el modo, el nombre y el tipo.