

UNIVERSIDAD CENTRAL DE LAS VILLAS

Facultad de Matemática–Física–Computación



TRABAJO DE DIPLOMA

GOSPEL

Ambiente para la programación de robots industriales

Autor: Dánel Sánchez Tarragó

Tutor: In. René González

“Año del 40 aniversario del triunfo de la Revolución”

1999

“Pero dentro de diez años Robots Universales Rossum producirá tanto trigo, tantos tejidos, tanto de todo, que las cosas prácticamente carecerán de valor. Cada cual podrá coger lo que quiera. No habrá pobreza. Sí habrá desempleo, pero no habrá empleo.

Todo lo harán las máquinas vivientes. Todo el mundo estará libre de preocupaciones y del degradante trabajo manual. Todos vivirán solo para perfeccionarse.”

Domin, primer acto de “R.U.R.”

Karel Capek

Índice

RESUMEN	4
SUMMARY	5
INTRODUCCIÓN	6
PROGRAMACIÓN DE ROBOTS	8
Lenguajes de programación de robots	9
Métodos de programación de robots	13
GOSPEL: SOFTWARE PARA LA PROGRAMACIÓN DE ROBOTS	15
DESCRIPCIÓN DEL SISTEMA.....	16
EL SISTEMA GOSPEL	17
EL LENGUAJE.....	19
CARACTERÍSTICAS DEL EDITOR DE PROGRAMAS.....	23
EL GOSPEL POR DENTRO.....	26
LOS PRODUCTORES	27
EL PRODUCTO.....	32
CONCLUSIONES Y RECOMENDACIONES	39
BIBLIOGRAFÍA.....	40
ANEXO 1 COMANDOS MÁS IMPORTANTES DEL EDITOR DE CÓDIGO	41
ANEXO 2 DESCRIPCIÓN DE LOS COMANDOS DEL MENÚ	43
ANEXO 3 INTRODUCCIÓN A LA NOTACIÓN UTILIZADA	45

Resumen

El objetivo del presente trabajo es desarrollar una aplicación que ofrezca un ambiente adecuado para la programación de robots industriales. Se diseñó un lenguaje para este propósito que permite la programación textual explícita, se construyó un compilador y un intérprete para éste, y se crearon bibliotecas de rutinas que enriquecen las potenciales del sistema. El producto fue acoplado a un ambiente gráfico que permite la simulación de las tareas y se le incorporaron utilidades como un editor de código y un módulo de depuración de programas.

Summary

The objective of the present work is to develop an application that it offers an appropriate environment for the programming of industrial robots. A language was designed for this purpose that allows the explicit textual programming, it was built a compiler and an interpreter for this, and libraries of routines were created that enrich the potentials of the system. The product was coupled to a graphic environment that allows the simulation of the tasks and there were incorporated utilities like a code editor and a module of purification of programs.

Introducción

La introducción de la robótica en Cuba comienza a mediados de la década del 80 con la adquisición de un robot de soldadura por una empresa industrial. Más tarde es creada una célula flexible por ICIMAF con fines docentes y se comienza la producción de un lote de robots industriales por EICISOFT, algunos de los cuales fueron instalados en la capital. Sin embargo, este breve período de desarrollo y esplendor se vio truncado como consecuencia de las difíciles condiciones económicas en que se sumió el país a finales de los '80.

Teniendo en cuenta el potencial tecnológico desarrollado en el área de la robótica industrial que actualmente se encuentra inutilizado y con tendencias a desaparecer el Grupo de Investigación de Robótica de la UCLV, en conjunto con el Centro de Investigación de Soldaduras (CIS), se trazó el objetivo de desarrollar una tecnología para la recuperación de las características de operación de los robots industriales. Esto permitirá desarrollar una base de operación con esta tecnología en la Universidad Central de Las Villas que garantizará la formación de los profesionales que eventualmente la necesiten, así como desarrollar tareas de soldadura aprovechando el trabajo conjunto que se realiza con el CIS de la UCLV, el que posee una diversidad de máquinas de soldadura automática y cuenta con una amplia experiencia en esta técnica.

Parte importante de las investigaciones en esta línea se ha llevado a cabo a través de los trabajos de diploma realizados por estudiantes de la Facultad de Ingeniería Eléctrica y de la Facultad de Mecánica. Como resultado hoy se cuenta con gran parte de la electrónica necesaria para el funcionamiento de los robots, se construyeron los *drivers* para los motores [NIUBÓ-97], se recuperaron las tarjetas de lectura de los *encoders* [ESCALONA-98] y se programaron los elementos básicos de control para manejar al robot desde una computadora.

El presente trabajo viene a continuar los esfuerzos que se han venido realizando en la parte del desarrollo del software necesario para controlar y programar los manipuladores. Con el objetivo fundamental de construir una herramienta CAD-CAM para la automatización del trabajo con robots industriales y posibilitar la simulación y control de las tareas se diseñó el lenguaje Gospel, basado en Pascal, pero que soporta un conjunto de rasgos que permiten su aplicación a la robótica. Se construyó un compilador para este lenguaje y varias herramientas de apoyo tales como un editor de programas y un módulo de depuración, y se acopló a un simulador gráfico 3D. Todo esto elevó en gran medida la calidad de la comunicación hombre-robot, facilitando el desarrollo de aplicaciones de robótica.

En este informe se dedicó un primer capítulo a exponer los principios básicos que caracterizan a los ambientes de programación de robots industriales con el fin de lograr una mejor comprensión del trabajo realizado. En un segundo apartado se explica sucintamente las características que presenta la aplicación desde el punto de vista del usuario, los elementos del lenguaje y la interfaz gráfica. Por último, en el tercer capítulo se analizan aspectos generales del diseño y la implementación del sistema.

Programación de robots

Según la definición adoptada por la RIA (Robot Industry Association), un robot industrial es un manipulador reprogramable multifuncional diseñado para mover materiales, piezas, herramientas o artefactos especiales mediante movimientos variables programados para la ejecución de tareas. En la industria se utilizan robots típicamente para la manipulación de materiales, alimentación de maquinaria, pintura y sellado, soldadura, mecanizado, ensamblado e inspección.

La propiedad más importante de los robots dentro de la automatización de un proceso, es su capacidad de ser reprogramados. Esto los hace flexibles y poderosos, porque se adaptan a las necesidades de producción sin que sea indispensable modificar físicamente las máquinas que elaboran el producto, como ocurre con la automatización rígida.

Esta aparente ventaja se puede convertir en una desventaja si la labor de programación se vuelve demasiado dispendiosa o si los costos de reprogramar el robot resultan muy altos. Si es necesario detener una línea completa de producción durante varias semanas para hacer y probar un programa de robot, su utilización pierde rentabilidad y deja de ser una solución atractiva. Los ambientes de programación de robots son un intento por resolver este problema. Para esto enfocan toda la ingeniería tradicional de software, pensando en los problemas específicos de la programación de robots y creando herramientas de apoyo.

Lenguajes de programación de robots

Un programa de robot es una especificación de una secuencia de movimientos y otras acciones individuales que debe realizar para ejecutar una tarea concreta. Un lenguaje de programación de robots es el formalismo sintáctico y semántico para especificar dicha secuencia. Por tanto, debe diseñarse desde el punto de vista de las tareas que podría realizar un robot. Esto condiciona el tipo de datos, instrucciones y funcionalidad que debe ofrecer al programador [BLUME–86].

Usualmente la tarea de un robot consiste en cambiar el estado de las cosas que le rodean (por ejemplo mover o ensamblar piezas, soldar o pintar objetos, etc.). Se denomina **entorno de trabajo del robot** o simplemente entorno, al conjunto de espacio y objetos que pueden ser alcanzados por el robot y estarán involucrados en la realización de las tareas.

Tanto el robot como su entorno pueden definirse a partir de un conjunto de variables que especifican su estado en términos del espacio que ocupan y la relación que tiene con otros objetos (por ejemplo está encima de, está insertado en, etc.).

Teniendo en cuenta estas definiciones se puede establecer que, en el nivel más abstracto, la labor de programación de un robot debe incluir las siguientes etapas:

1. Definir el resultado de la tarea (post-condición): debe ser claro cual es el resultado deseado después de ejecutar el programa expresado en términos del estado del robot y su entorno.
2. Definir el estado inicial del robot y el entorno de trabajo (pre-condición): se debe establecer el estado que deben tener el robot y los demás objetos que cambiarán su estado durante la ejecución de la tarea.

3. Desarrollar una estrategia para alcanzar el resultado, partiendo del estado inicial.
4. Establecer el conjunto de movimientos y acciones del robot que componen la estrategia.

Desde el punto de vista funcional, un lenguaje de programación de robots debe proveer un conjunto de elementos de diferentes tipos que permitan realizar las siguientes funciones:

Control de movimiento y actuación

Esta es una de las funciones principales que debe ofrecer un lenguaje de programación de robots. Dentro de este apartado debe haber instrucciones que permitan especificar posiciones que deban ser alcanzadas por el extremo del robot y la forma de llegar a ellas: moviendo los ejes independientemente, moviendo todos los ejes simultáneamente (movimiento coordinado) o siguiendo una trayectoria establecida (línea recta, arco de circunferencia, etc.). Adicionalmente es necesario poder determinar la velocidad y aceleración del movimiento o el tiempo requerido para realizar el movimiento. Otro elemento interesante es poder determinar si una posición debe alcanzarse con mucha precisión o solamente es un punto de paso dentro de una trayectoria.

Otras funciones deseables incluyen el control de fuerza o par aplicado por el extremo del robot, el control del efector final (herramienta, pinza, etc.) y de aparatos auxiliares (por ejemplo equipos de soldadura) y la coordinación de movimientos con otros robots o máquinas que actúen en el mismo entorno.

Control de información sobre el entorno

En este apartado se incluyen funciones que permitan la comunicación del robot con los demás elementos del entorno. Esta comunicación generalmente se realiza por medio de sensores externos que se ubican sobre el robot o en puntos estratégicos dentro del entorno.

Este conjunto de funciones es de gran utilidad para iniciar o detener la ejecución de programas, seleccionar acciones alternativas, identificar y posicionar objetos o rasgos del entorno (por ejemplo ubicar el borde de la mesa), realizar movimientos protegidos (por ejemplo moverse hasta encontrar la pared) o establecer restricciones acomodaticias (por ejemplo insertar mientras la fuerza no supere 100 newtons).

Control y supervisión del sistema

Finalmente se considera un conjunto de funciones utilitarias de supervisión y actuación del sistema mismo que generalmente son transparentes para el programador. Entre estas se cuentan: supervisión del control del movimiento, interfaz con los periféricos externos (discos, monitores, red, etc.), proceso y almacenamiento de datos (constantes definidas, valores de variables, subrutinas, etc.) y otras funciones internas de supervisión (control de interpretación, compilación, etc.).

De acuerdo con las funciones comentadas anteriormente se puede establecer el conjunto de elementos generales (tipos de datos, instrucciones y software auxiliar) que debe tener un lenguaje de programación de robots. A continuación se describen algunos de ellos haciendo la comparación con los lenguajes de programación de computadoras.

En el manejo de datos deben tenerse en cuenta los datos escalares y otros datos específicos para robots como son: datos de tipo posición-orientación del extremo, sistemas de coordenadas, tipo de trayectoria, etc. Es entonces necesario que el conjunto de instrucciones del lenguaje permita manejar estos datos específicos, además de controlar el movimiento del robot, manejar las entradas y salidas (digitales y analógicas) y controlar el flujo del programa mediante condicionales (si entonces, en caso de, etc.) y repetición de bucles (mientras que, repita n veces, etc.).

Por otra parte el lenguaje debe estar contenido dentro de un ambiente más amplio de programación con un mínimo de utilidades tales como un editor de programas, posibilidades de depuración de programas, manejo de archivos, un interprete o compilador y eventualmente un simulador gráfico. Algunas capacidades más “inteligentes”, como son la representación de formas de conocimiento, el modelado del entorno, la especificación de tareas y la toma de decisiones son muy deseables pero aún se encuentran en fase de desarrollo y experimentación.

Los lenguajes de programación que se encuentran en el mercado, ofrecen muchas de las funciones enumeradas anteriormente. Hoy en día no es extraño encontrar robots con adiciones especiales para soldadura capaces de seguir automáticamente un cordón y regular variables como la corriente y la velocidad de avance, de tal forma que se garantice una buena calidad de la unión soldada. Sin embargo, otros elementos con mayor potencialidad están hasta ahora en etapa de desarrollo y experimentación.

En general, la compra de un robot incluye el lenguaje de programación. Algunos ejemplos de lenguajes de programación de robots que se encuentran comercialmente, se enumeran en la siguiente tabla.

Nombre del lenguaje	Empresa que lo distribuye
ARLA, RAPID	ASEA BROWN BOVERI
VAL-II	UNIMATION
V+	ADEPT
AML / 2	IBM

Aunque teóricamente existen lenguajes adaptables a robots de diversos fabricantes, la práctica más generalizada es programar el robot con el lenguaje que ofrece el fabricante. Solo en algunos desarrollos industriales y en muchos desarrollos de investigación se descarta el lenguaje vendido por el fabricante y se reemplaza por lenguajes diseñados a medida.

Métodos de programación de robots

La principal funcionalidad de un programa de robot es especificar una serie de posiciones y orientaciones que debe asumir el extremo del robot para recorrer trayectorias que le permitan cumplir con las tareas deseadas. Adicionalmente, se intercalan instrucciones de actuación de la herramienta, pinza o actuador final que porta el robot en el extremo. Una vez que se especifica el programa, el robot utiliza algún algoritmo de control para enviar a los motores (o cilindros) que mueven las articulaciones las señales adecuadas para lograr los movimientos que permitan alcanzar las posiciones especificadas.

La programación de un robot se puede hacer en línea, utilizando directamente el robot, o fuera de línea, utilizando herramientas CAD, simuladores gráficos y utilidades de validación de programas. Por costo, comodidad y seguridad es más conveniente la última opción.

Desde la perspectiva del programador, existen diferentes niveles de abstracción en los cuales él puede expresar la tarea que debe ejecutar el robot: desde la especificación física de cada posición, hasta la formulación de la tarea a través de dos situaciones una inicial y otra final sin indicar de manera explícita la forma de pasar de una a la otra. Estas dos formas definen el espectro de posibilidades que se utilizan para programar robots: programación gestual y programación implícita [BARRIENTOS-97]. Además de estas dos, existe una manera intermedia conocida como programación textual explícita, equivalente de cierta forma a la programación imperativa en informática.

En la **programación gestual** el programador guía físicamente al robot para enseñarle la manera de ejecutar la tarea, el robot memoriza las posiciones y movimientos y luego los repite. Es la manera más fácil y económica de programar un robot, pero tiene muchas limitaciones. El guiado se puede realizar por varios medios:

- Moviendo un maniquí y usando un modo de grabación.

- Mediante una paleta de programación (joystick y menús de instrucciones).
- Teleoperando el robot con una estrategia maestro esclavo.

En la **programación textual explícita**, el programador utiliza un lenguaje como VAL-II, V+ o LM con instrucciones de movimiento y, valiéndose de un compilador, genera las posiciones físicas que envía al robot. Es más general y poderosa que la programación gestual, pero todavía resulta insuficiente. Tiene la ventaja de ser más apropiada para realizarse fuera de línea (sin interrumpir el proceso de producción). También puede incluir la programación de otros sistemas que interactúen con el robot (por ejemplo bandas transportadoras, mesas posicionadoras o máquinas de control numérico).

Existen muchos lenguajes de programación explícita de robots [BLUME-86] y su utilización se encuentra bastante difundida. El siguiente es un fragmento de programa escrito en VAL-II, que toma una pieza y la lleva a otra posición, y puede dar una idea del tipo de instrucciones que componen estos lenguajes:

```
MOVE #PPOINT(0, 90, 90, 60, 45, 30)
APPRO #POS1, -100
OPENI 50
DEPARTS -100
CLOSEI 20
MOVE #POS2
```

La **programación implícita**, por su parte es el equivalente a la programación automática en informática y todavía se encuentra en fase de investigación. El programador especifica, con algún formalismo, la tarea que se desea resolver (con dos escenas, por ejemplo), y el sistema, con algunas pautas generales y la descripción cinemática y geométrica del robot, genera las instrucciones de movimiento que resuelven la tarea.

Esta última forma de programar es muy poderosa, porque es independiente del robot, fácil de mantener, rápida, etc., pero tiene varios problemas que no se han

logrado resolver todavía de forma satisfactoria, sobre todo en lo concerniente a la eficiencia. Existen versiones comerciales parciales de estos lenguajes (AUTOPASS, LAMA) pero, debido a las grandes restricciones que tienen, son muy poco utilizados comercialmente. El siguiente es un fragmento de código de AUTOPASS para ensamblar dos objetos y colocarlos sobre otro: especifica tres tareas, pero no se refiere en ningún momento a las posiciones que debe alcanzar el robot para resolverlas:

```
GRASP OBJETO1  
INSERT OBJETO1 INTO OBJETO2  
PLACE OBJETO2 ON OBJETO 3
```

Gospel: software para la programación de robots

El resultado del presente trabajo de diploma ha sido la construcción del sistema Gospel. Este software tiene como propósito crear un entorno de programación adecuado para el desarrollo de aplicaciones industriales de robótica. En su realización se ha tenido en cuenta, dentro de las posibilidades reales, todos los aspectos mencionados en esta sección.

Gospel permite realizar la programación a través de dos de los métodos anteriormente expuestos: el método de programación gestual y el de programación textual explícita. La programación textual se ejecuta por medio de un lenguaje específico diseñado para este propósito.

El lenguaje ha sido dotado con un conjunto de instrucciones para el movimiento, actuación y control de los mecanismos, y se han añadido herramientas que enriquecen el entorno de programación.

Descripción del sistema

En este capítulo se hará una exposición de las características más notables de la aplicación desde el punto de vista de un usuario.

Note que los usuarios del sistema Gospel son, al mismo tiempo, desarrolladores de programas. Sin embargo, esta aplicación ha sido diseñada teniendo en cuenta que quien estará realizando los programas para la manipulación del robot no es un especialista en ciencias de la computación, sino un tecnólogo con limitadas habilidades en programación.

Por este motivo se dedicaron esfuerzos para lograr una interfaz de usuario que fuera sencilla, agradable e intuitiva. Por otro lado se decidió basar el lenguaje en la gramática de Pascal, se buscó simplicidad en los tipos de datos y, en general, se trató de liberar al programador de la carga de conocer pormenores de la arquitectura de las máquinas y otros detalles.

Se dará primeramente una visión general de todo el sistema, luego se mostrará los elementos del lenguaje haciendo énfasis en las particularidades que se aplican a la robótica y se terminará haciendo una breve descripción del editor de programas y su ambiente de desarrollo.

El sistema Gospel

El Gospel constituye un ambiente de programación para manipuladores. Incluye un conjunto de facilidades que permiten la programación de un robot fuera de línea, utilizando herramientas CAD, un simulador gráfico y mecanismos de validación de programas.

Como resultado del trabajo de investigación desarrollado por Grupo de Investigaciones Mecatrónicas [ESCALONA-98] se obtuvo una primera aplicación que permitía la programación gestual e incluía un simulador gráfico para examinar las trayectorias generadas. Actualmente esta aplicación cuenta con nuevas herramientas, fruto de la presente tesis, que han permitido elevar el nivel de la programación desde el método por guiado a la programación textual explícita.

Para esto se desarrollaron utilidades tales como un editor de programas, un módulo de depuración de programas, manejo de archivos, un compilador e intérprete, interfaz con AutoCAD, entre otras.

La aplicación presenta en su pantalla principal un puerto gráfico 3D interactivo implementado con herramientas DirectX [MSDN-98] que no solo es utilizado por el simulador para mostrar las distintas trayectorias generadas, sino también permite comunicarse con el sistema de guiado indicando las posiciones del manipulador tan solo con movimientos del ratón.

En el puerto gráfico se muestra el manipulador en su posición actual y, además, pueden ser ubicados objetos de distintos tipos para representar el entorno y analizar posibles colisiones con obstáculos cercanos.

En primera pantalla aparece también el panel de los parámetros de entrada que permite al usuario que realiza una programación por guiado especificar de forma exacta los datos en forma de posición del instrumento o como ángulos de las articulaciones.

El editor de programas junto con el compilador y las demás opciones del ambiente de programación textual ha sido integrado a la aplicación principal y puede ser accedido a través del comando Editor en el menú.

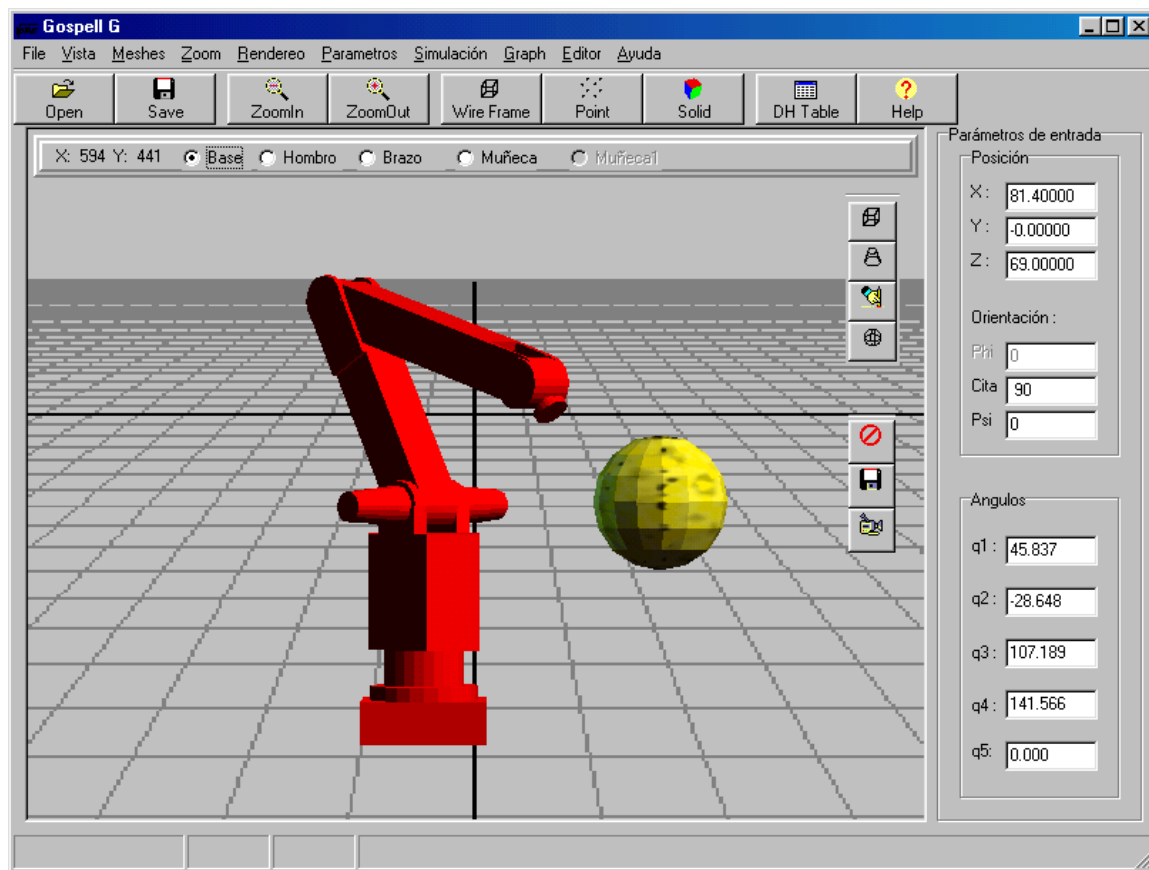


Figura 1 Pantalla principal de la aplicación.

Una vez escrito el programa en el editor de código basta mandar a ejecutarlo para observar en la pantalla del simulador la trayectoria generada por este. Si la trayectoria no es del agrado del tecnólogo se puede retornar al editor para hacerle las correcciones necesarias. Este proceso podrá repetirse hasta tanto no se esté satisfecho con la ejecución realizada. Cuando se obtenga la trayectoria definitiva esta podrá ser salvada para ser luego ejecutada por el robot.

El lenguaje

Con igual nombre que la aplicación que lo implementa, éste constituye un lenguaje de programación textual de alto nivel que incluye algunas características importantes, tales como la utilización de funciones y procedimientos, la posibilidad de declarar rutinas y datos como locales o globales y una estructura completamente modular del programa.

Es un lenguaje altamente estructurado a manera de un lenguaje de programación de propósito general. Las aplicaciones desarrolladas en Gospel incluyen el programa en sí y módulos pertenecientes a las bibliotecas del sistema que contienen rutinas y datos de tipo general, independientes del programa pero que pueden ser utilizado por él en cualquier momento. A su vez el programa puede ser dividido en varios módulos, uno de los cuales ha de ser el principal.

Gospel implementa la programación explícita, aunque admite que en un futuro se desarrollen las herramientas necesarias para incluir comandos de movimiento que lleven implícito las coordenadas de las posiciones a alcanzar.

Las instrucciones generales de movimiento se definen basándose en cómo posicionar el punto central de la herramienta (TCP). Los movimientos se programan a partir de posiciones definidas, es decir, se le dice al robot que se mueva desde donde se encuentra a una posición determinada. En estos momentos se cuenta con las siguientes instrucciones de movimiento:

Park

MoveAng ($\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$)

MoveCar ($x, y, z, \alpha, \beta, \gamma$)

Estas son instrucciones de movimiento simples, es decir, son comandos que solo especifican la dirección del destino final y no llevan parámetros como posiciones intermedias a alcanzar, velocidad o duración del movimiento.

La instrucción Park se utiliza para aparcar el manipulador, es decir, llevarlo a su posición inicial para calibrarlo.

La instrucción MoveAng recibe como parámetro coordenadas articulares o del robot, es decir, los ángulos que deben girar los motores de cada una de las articulaciones del robot para posicionar y orientar su extremo como el objeto en cuestión. Por ejemplo:

MoveAng (0, 45, 0, 0, 0)

Ordena a la articulación número dos que gire un ángulo de 45° en el sentido positivo.

Sin embargo, una especificación en coordenadas cartesianas es mucho más cómoda para la comprensión humana, por ello a la instrucción MoveCar se le pasan seis parámetros, los tres primeros son la posición del extremo efector asociada a un sistema de referencia y los tres últimos su orientación en ángulos de Euler.

Además de las mencionadas anteriormente existe la instrucción Interpol que si bien no mueve directamente al manipulador si permite generar una trayectoria a partir de determinados parámetros que se le especifican [Ballesteros–89].

Interpol (Po, Pf, N, Met)

Esta instrucción construye una trayectoria que comienza en el punto Po y termina en Pf interpolando a través de N puntos intermedios usando el método indicado por el parámetro Met. De esta manera se pueden realizar movimientos más complejos no solo siguiendo líneas rectas sino también en forma de arco de círculo, de parábola u otras curvas.

En un futuro, a medida que se vayan diseñando nuevos algoritmos de control, podrán implementarse otras instrucciones que incluyan parámetros de control del movimiento como puede ser la velocidad, la aceleración, la duración, la precisión, entre otros.

Entre las estructuras sintácticas el lenguaje cuenta con dos tipos de rutina posibles:

- **Procedimientos** : Rutina que no devuelven ningún valor y que se utiliza como una instrucción.
- **Funciones** : Rutina que devuelve un dato de tipo específico y que se utiliza como una expresión.

Los datos a manejar pueden ser de dos tipos:

- **Dato atómico**: No se define en función de otro tipo y no se puede dividir en diferentes componentes.
- **Dato estructurado**: Está formado por una serie de componentes con sus respectivos nombres. Los componentes pueden ser a su vez de tipo atómico o de tipo estructurado.

Estos datos, que se pueden definir como globales en módulos o locales en subrutinas, pueden a su vez ser definidos como constantes o variables. Los tipos básicos que soporta son: Integer, subrango, enumerado, Real y String. Mientras que entre los tipos estructurados están los registros (record) y los arreglos (array).

Además de los datos convencionales el sistema Gospel cuenta con otros específicamente destinados a definir las operaciones de interacción con el entorno, como son por ejemplo, los que especifican la posición y orientación de los puntos y objetos a los que debe acceder el robot (Frame).

Hay que destacar que en robótica el uso de las matrices resulta imprescindible. Todas las transformaciones de traslación, giro, cambio de ejes, etc. se plantean mediante un enfoque matricial [USÁTEGUI-90], así mismo los problemas de cinemática y los dinámicos se describen con el uso de matrices. Teniendo en cuenta estos factores se le ha prestado gran consideración a este tipo de datos. El lenguaje incluye vectores (ColVector y RowVector) y matrices (Matrix), y define el conjunto estándar de operaciones matriciales.

En la representación de los vectores y las matrices, así como de las operaciones entre ellos, se usó el formato de MatLab debido a que este es el paquete de cálculo utilizado por excelencia por los ingenieros.

Se cuenta, además, con las operaciones aritméticas habituales (asignación, +, -, *, /) que se pueden asociar tanto a números como a vectores o matrices, con algunas funciones matemáticas particulares como son *Sin*, *Cos*, *ArcTan2*, entre otras, y operadores booleanos para el tratamiento de variables de tipo boolean.

No podemos dejar de mencionar entre los tipos de datos del lenguaje Gospel al tipo DXF debido a las potencialidades que brinda. El tipo DXF representa a un archivo de AutoCAD en el formato homónimo. Se le pueden aplicar varias operaciones de transformación como escalado, traslación, etc. y acceder a las distintas entidades del dibujo. Esto permitiría al especialista hacer un plano en AutoCAD con todas las posibilidades que este brinda para el diseño gráfico, especificar ahí mismo las trayectorias a seguir y luego solo habría que importar el archivo hacia una variable de tipo DXF, identificar los recorridos y mandarlos a ejecutar.

El sistema de programación Gospel posee una variada gama de instrucciones para controlar el flujo de ejecución del programa, entre las que se destacan: llamadas a rutinas con distintos parámetros, instrucciones condicionales, instrucciones de repetición tipo FOR, tipo UNTIL o tipo WHILE. La sintaxis de estas estructuras gramaticales es similar a la de Pascal.

Una parte fundamental de la potencialidad del lenguaje la constituye las rutinas predefinidas que conforman la BRL. Esto es un conjunto de procedimientos y funciones construidos dentro del compilador que brindan una serie de utilidades de diversos tipos. Por ejemplo, las rutinas de procesamiento de imágenes permiten aplicar varias operaciones y hacer algunas transformaciones sencillas como llevar a tonos de grises, umbralizar imágenes, etc. Otras rutinas se orientan a las funciones de movimiento, visión y prestaciones matemáticas.

Características del editor de programas

Como en cualquier lenguaje de programación convencional, es de gran importancia, para conseguir un aumento considerable de la productividad de la programación, el contar con un entorno de desarrollo adecuado a las necesidades.

Partiendo de estas premisas, se concibió lo que se denominó el salón de programación, donde el usuario cuenta con los útiles necesarios para construir sus aplicaciones.

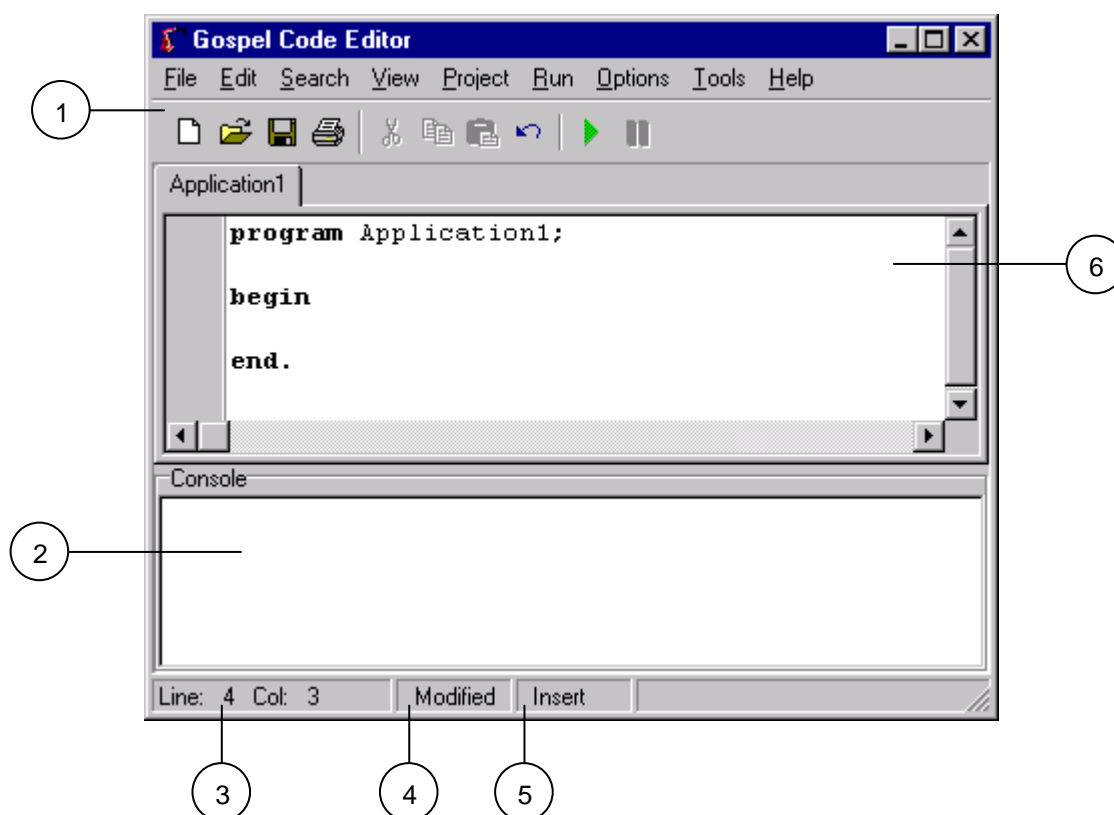


Figura 2 Apariencia del salón de programación.

A continuación se describen los elementos visuales señalados:

1. Barra contenedora de los botones de aceleración. Estos botones permiten acceder de forma más rápida a los comandos más usados del menú.
2. Consola de salida. Es una sencilla interfaz de salida que permite al programador imprimir mensajes a través de la instrucción Output del Gospel.
3. Muestra la posición actual del cursor.
4. Indica si el texto de la página activa en el editor de código ha sido modificado desde la última vez que fue salvado.
5. Indica si el editor se encuentra en modo de inserción o de sobrescritura.
6. Editor de programas. Permite ver y modificar cualquier parte del código fuente contenido en la página activa.

El editor de programas brinda acceso al código que ejecuta la aplicación y presenta algunas de las más importantes características para la edición como son:

- Sintaxis resaltada en colores
- Deshacer y rehacer múltiples comandos de edición
- Una amplia gama de comandos de edición

Para ver una lista con los comandos de edición más importantes puede remitirse al anexo 1.

Usted puede abrir múltiples archivos en el editor de código. Cada archivo que se abre es una nueva página del editor, y cada página es representada por una etiqueta encima de la ventana. Cuando se abre una página del editor de código, usted puede explorar todos los datos que contiene, no sólo secciones particulares de su código.

El editor de programas presenta las siguientes opciones que se pueden acceder a través del comando de menú Options | Preferences:

Auto Indent Mode: Pone el cursor bajo el primer carácter distinto de blanco de la línea no vacía precedente cuando se ejecuta un cambio de línea.

Insert Mode: Inserta el texto en la posición del cursor sin sobrescribir el existente. Si Insert Mode no está habilitado, el texto en la posición del cursor es sobrescrito. La tecla INS conmuta estas dos variantes.

Smart Tab: Tabula hasta el primer carácter distinto de espacio en la línea precedente.

Backspace Unindents: Alinea el punto de inserción con el nivel previo de sangría cuando se presiona Backspace si el cursor se encuentra en el primer carácter distinto de blanco de la línea.

En el anexo 2 puede consultar los comandos de edición más importantes.

El Gospel por dentro

En este capítulo se abordará lo referente al diseño del sistema Gospel, así como algunos aspectos de su implementación.

Hoy en día el Gospel es una herramienta aplicable para algunos desarrollos de robótica, sin embargo debemos recalcar que lo hecho hasta ahora constituye solo un cimiento para construir un software que cuente con todas las posibilidades para programar cualquier célula flexible. Teniendo en cuenta estas premisas se realizó el diseño de Gospel buscando sencillez y extensibilidad.

La aplicación fue programada en Delphi 4.0 siguiendo el paradigma de la programación orientada a objetos.

Para un mejor estudio del diseño del sistema, se ha hecho una división lógica en dos dominios:

- **Productor** : donde se encuentran las clases que juegan un papel activo en el tratamiento de la información.
- **Producto** : son las clases que conforman el código interno del lenguaje.

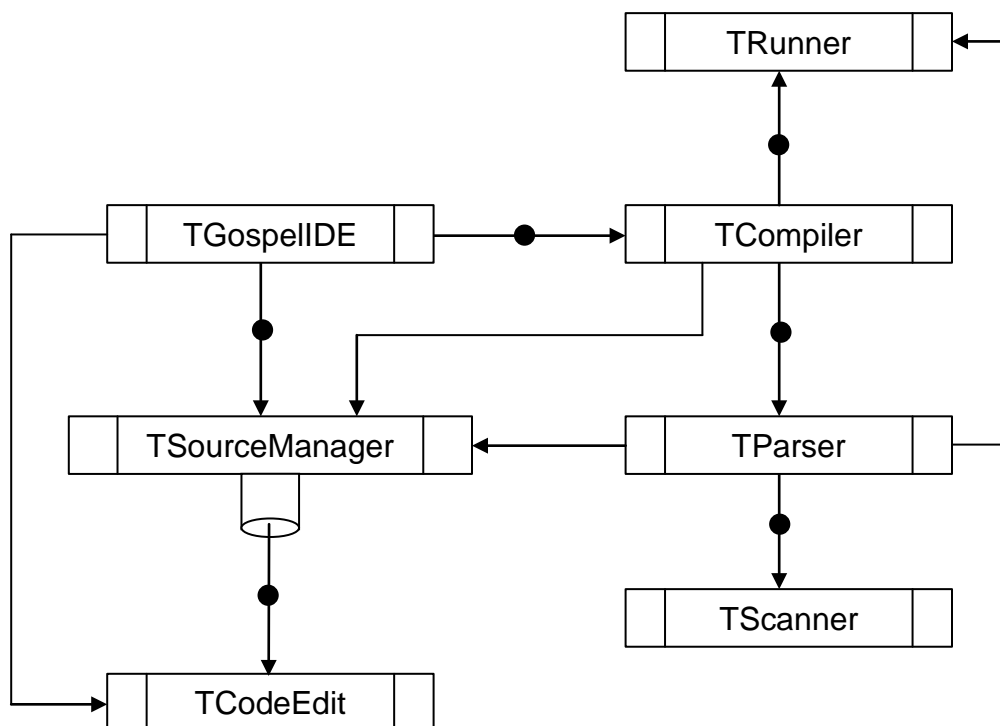
A continuación se hace un análisis general de cada uno de estos dominios mostrando las relaciones entre las entidades fundamentales y describiendo brevemente las clases integrantes. La notación utilizada para los diagramas se explica en el anexo 3.

Los productores

Este dominio está integrado por las clases encargadas de analizar y procesar la información suministrada por el usuario y entregar como resultado un producto semielaborado que es el código interno. Luego éste es ejecutado y se convierte a su vez en código entendible por el sistema simulador o por el robot.

Realmente este grupo de clases, más que producir, transforman la información; y una entrada de un nivel basto es convertida en una salida superior, más aguda y organizada. Sin embargo, se decidió denominarlos productores –y no transformadores– por hacer un contraste con las clases que forman el código interno y que representan el producto de estas transformaciones.

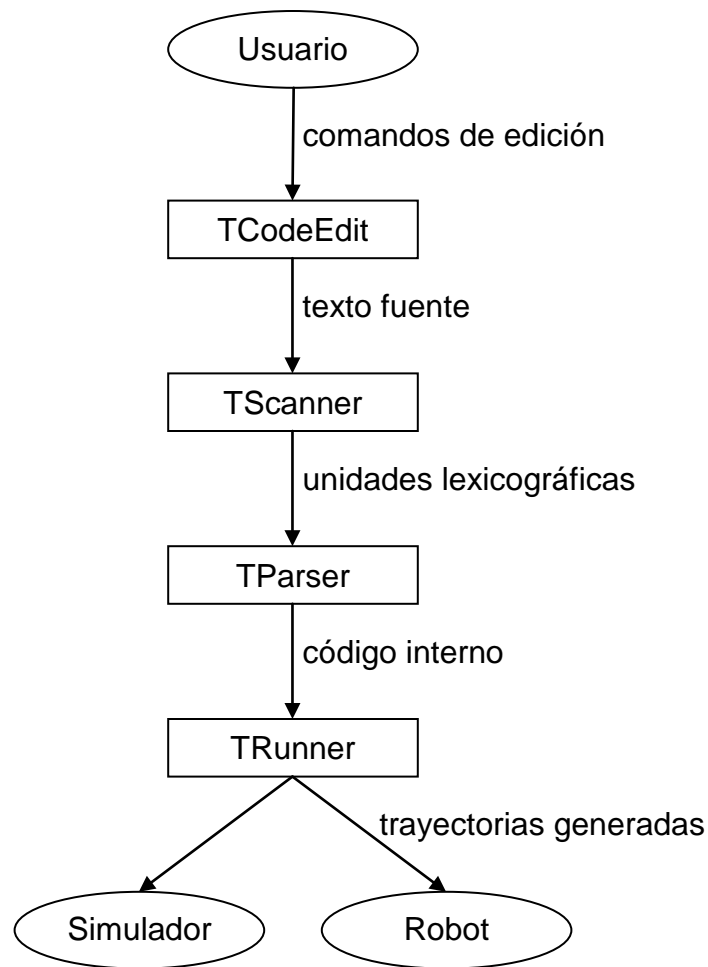
El diagrama que aparece a continuación revela la relación que existe entre las distintas clases que conforman este dominio.



No todas las clases que se encuentran en este dominio son transformadoras de información. Las clases TGospelIDE, TSourceManager y TCompiler tienen una

función controladora: interactúan con el resto de las clases y determinan la forma de manejar los datos.

El siguiente esquema muestra cómo ocurre el flujo de la información entre las distintas entidades que integran esta parte del sistema en el proceso de compilación y ejecución de un programa. Las flechas indican la dirección en que se realiza la comunicación.



A continuación se describen someramente cada una de las clases mencionadas en este apartado.

TGospelIDE

Principal clase de la interfaz de usuario. Permite la comunicación del usuario con cada uno de los componentes del sistema. Contiene a todas las clases visuales y manipula todos los eventos externos. Es TGospelIDE quien construye, y al final destruye, las instancias de TCompiler y de TSourceManager.

TCodeEdit

Es la clase visual que permite la edición del código fuente del programa. TCodeEdit hereda de TmwCustomEdit la cual define sus propiedades básicas. Esta última clase fue extendida a partir del código original de Martin Waldenburg y sus colaboradores y se le añadieron características como indentación, tabulación inteligente, locación de puntos de ruptura, entre otras. TCodeEdit tiene una referencia a una instancia de la clase TmwCustomHighLighter que es la responsable de resaltar la sintaxis.

TSourceManager

Es el administrador de las fuentes del proyecto. Contiene una lista de instancias de TCodeEdit, permitiendo el acceso a las características de la edición. Tiene toda una serie de métodos para la manipulación de los archivos del proyecto (*NewApplication*, *OpenFile*, *SaveModule*, etc.), así como distintos atributos propios del proyecto y su entorno (*SearchPath*, *AppDirectory*, etc.).

TScanner

Es la clase encargada de extraer y clasificar cada uno de las unidades básicas lexicográficas de las fuentes. Estas unidades pueden ser números enteros, números reales, cadenas de caracteres, palabras claves, comentarios o símbolos. El texto que se está analizando puede ser accedido a través de la propiedad *Source*, mientras que los atributos *Token*, *Position* y *TokenType*, se refieren a la última unidad lexicográfica reconocida, su posición en el texto y su clasificación respectivamente.

TParser

Realiza el análisis sintáctico y semántico del código fuente [KATRIB-83] y a partir de este genera el código interno. Se diseñó un analizador descendente recursivo de una sola pasada con el objetivo de ganar en sencillez y claridad para futuras ampliaciones del lenguaje. Contiene una instancia de TScanner, el cual va a proveer a TParser de unidades lexicográficas que obtiene a partir del archivo fuente. Posee un conjunto de métodos que le permiten reconocer las distintas estructuras gramaticales del lenguaje.

TCompiler

Es la clase encargada de supervisar la conversión del código fuente en código interno, así como la ejecución de este último. TCompiler contiene una instancia de TParser con el objetivo de convertir los archivos fuentes en módulos compilados. Para acceder a las fuentes se auxilia de su relación con TSourceManager. El módulo principal, que se obtiene como resultado de la compilación del programa principal se accede a través de la propiedad *MainModule* de tipo TModule. Contiene también una instancia de TRunner, la cual es necesaria en el proceso de compilación.

TRunner

Su función es controlar la ejecución del programa. Esta clase encapsula un hilo de ejecución –paralelo al hilo de la VCL– en el cual corre las sentencias del programa. (Se utiliza la clase TAxIThread, desarrollada por Medardo Rodríguez, a quien le agradecemos profusamente su colaboración.) TRunner manda a cada una de las sentencias que conforman el código del programa a ejecutarse en el orden correspondiente teniendo en cuenta las condiciones de ejecución (programa corriendo, programa pausado, punto de ruptura, etc.). Para este propósito tiene la propiedad *StartPoint*: TStatement, que indica la sentencia que da comienzo al programa y el método procedure Run(Statement: TStatement); encargado de comandar la ejecución de la sentencia que se le pasa. También

cuenta con una serie de métodos (*TraceInto*, *StepOver*, *Pause*, *Reset*), así como propiedades y eventos para controlar la ejecución y la puesta a punto.

Interfaz de control

El sistema Gospel no es resultado del trabajo de una sola persona, sino de un equipo de investigación (GIMAS [G+]) integrado por programadores y técnicos de distintas esferas. Los distintos algoritmos de control empleados en las instrucciones son implementados por ingenieros en control automático y constituyen un módulo aparte en el diseño del sistema.

Es por tanto un objetivo importante para la arquitectura del compilador de Gospel mantener una interfaz mínima con las estructuras que llevan a cabo los algoritmos de control, de manera que permita una comunicación efectiva sin importar los cambios que se puedan realizar a cada lado de la misma.

Para este propósito se creó la clase *TMainHandler* cuyos métodos constituyen procedimientos de retro-llamada a los referidos algoritmos de control. Esta clase es instanciada en la aplicación como una variable global con el nombre *MainHandler*.

TSymbolTableEntry

Define una entrada en una tabla de símbolos. A partir de esta clase se especializan TDataType, TData y TBlock, que representan respectivamente los tipos; las variables y constantes; y los procedimientos, funciones y módulos del lenguaje. Por tanto cada uno de estas entidades tiene la capacidad de ser una entrada en una tabla de símbolos. Posee el atributo *Name* de tipo *string* que significa el identificador del símbolo.

TSymbolTable

Representa una tabla de símbolos. Esta clase mantiene una lista de instancias de TSymbolTableEntry y provee métodos para buscarlas por su nombre, accederlas o adicionarlas, entre otros.

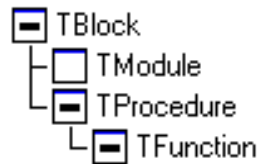
TSymbolSystem

Constituye un sistema de tablas de símbolos. Esta clase posee cinco tablas de símbolos para registrar por separado cada tipo de entidad del lenguaje (tipos, variables, constantes, procedimientos y funciones) con el objetivo de ganar en eficiencia de almacenamiento y velocidad de recuperación.

TBlock

Define el comportamiento básico de las estructuras de bloque que existen en el lenguaje. Los bloques pueden contener a otros bloques, es por ello que tienen una propiedad llamada *Owner* de tipo TBlock. Como cada bloque contiene una instancia de TSymbolSystem, se forma, al encajar un bloque en otro, un sistema general de tablas de símbolos con estructura jerárquica con una consecuente mejora en su rendimiento.

A partir de TBlock se especializan las siguientes clases:



TModule: Encapsula las propiedades y comportamiento de un módulo compilado.

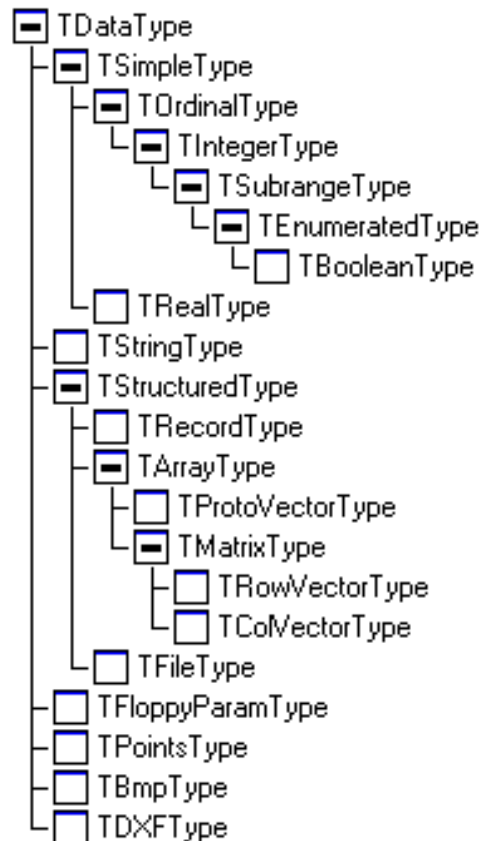
TProcedure: Esta clase representa a un procedimiento del lenguaje. Se puede heredar de ella para extender su comportamiento, por este medio se implementan las rutinas predefinidas que aparecen en la BRL.

TFunction: Representa a una función del lenguaje. Se puede heredar de ella para extender su comportamiento.

TDataType

Es la clase base de todos los tipos de datos primarios del lenguaje. Tiene un método nombrado *Instantiate* que se llama para construir una instancia de TData.

A partir de TDataType heredan las siguientes clases:



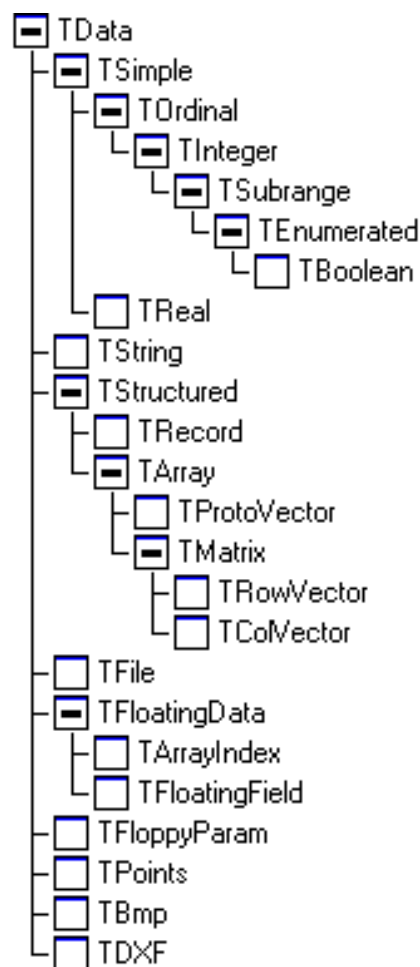
Estas clases conforman el repertorio de tipos primarios del lenguaje a partir de los cuales se pueden construir tipos más complejos.

TData

Es la clase base de los datos de tipo primario del lenguaje. Cuando un dato es creado a través del método *Instantiate* de una de las clases que especializan a TDataType, TData establece una referencia a la clase que lo creó. Con este mecanismo se garantiza que se puedan crear datos a partir de nuevos tipos declarados por el usuario.

En TData se declara una serie de métodos para realizar distintas operaciones sobre los datos; cada una de las clases que hereden de TData deberán redefinirlos.

A partir de TData heredan las siguientes clases:

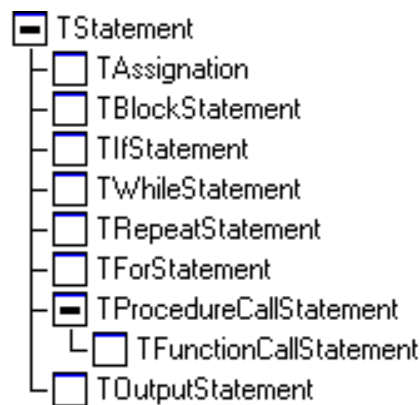


TStatement

Define el comportamiento básico de todas las sentencias del lenguaje. Las clases que heredan de TStatement redefinen el método *Execute* para especificar de qué forma se va a ejecutar esta sentencia. Aunque cada sentencia sabe cómo se va a ejecutar, ésta deberá hacerlo a través del objeto TRunner ya que es quien lleva el control de la ejecución del programa. Es por ello que TStatement mantiene una referencia a una instancia de TRunner.

TStatement tiene otras propiedades como *SourceLine: integer* y *HasBreakpoint: Boolean* para propósitos de puesta a punto.

A partir de TStatement se especializan las siguientes clases:

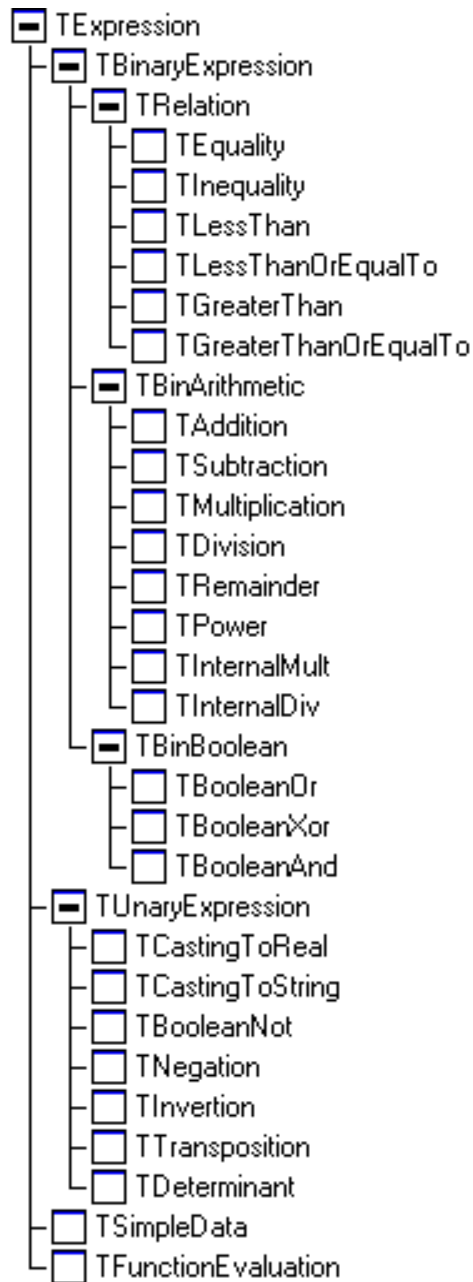


TExpression

Es la clase a partir de la cual se especializan todos los posibles tipos de expresiones que admite el lenguaje. Las sentencias de asignación, las condicionales y los parámetros de las rutinas utilizan instancias de TExpression. El método *Evaluate* retorna un TDate con el valor de la expresión, mientras que la propiedad *YieldType* permite conocer de qué tipo es la expresión.

A partir de TExpression se derivan dos líneas de herencia fundamentales: TBinaryExpression que define una acción entre dos operandos que son a su vez expresiones y TUnaryExpression que representa una operación sobre una única expresión.

A continuación se muestra el árbol de herencia que conforma la totalidad de las expresiones representables en Gospel:



Conclusiones y recomendaciones

Podemos sacar en conclusión que los lenguajes de programación de alto nivel como Gospel proporcionan una mayor interacción entre el hombre y el robot. Además, presentan claras ventajas como son:

- **Inteligibilidad:** Se puede realizar una buena documentación, así como un diseño ordenado y coherente del programa.
- **Fiabilidad:** Sobre todo en sistemas que deban responder a situaciones imprevistas.
- **Adaptabilidad:** Los programas permiten modificaciones, mejoras y ampliaciones con, relativamente, poco esfuerzo.

En esta fase de trabajo se logró una herramienta CAD-CAM que posibilita la programación, control y simulación de tareas en robots industriales. Al mismo tiempo, la aplicación desarrollada constituye un medio importante de enseñanza para las asignaturas Robótica y Sistemas Flexibles en la maestría de Automática.

El mayor logro de este trabajo es, sin lugar a dudas, el hecho de haber sentado las bases para el despegue de la robótica en la Universidad Central de Las Villas. Se crearon varias herramientas de trabajo, se concibió un lenguaje con características específicas, aunque todavía quedan muchas cosas por hacer y en cuanto a esto hacemos las siguientes recomendaciones:

- Aumentar el repertorio de instrucciones del lenguaje destinadas al control de movimientos, sensores y visión.
- Introducir nuevas estructuras y mecanismos del lenguaje que permitan el método de programación orientada a tareas.
- Hacer un editor 3D que permita diseñar el entorno y los manipuladores dentro del ambiente de trabajo de forma interactiva.

Bibliografía

[BALLESTERO–89] Ballestero, R., L. Hernández: "Generación de trayectorias." Informe final de investigación. UCLV Cuba. 1989.

[BARRIENTOS–97] Barrientos Antonio. "Fundamentos de Robótica." España 1997.

[BLUME–86] Blume, C., W. Jakob: "Programming languages for industrial robots." Springer-Verlag 1986.

[ESCALONA–98] Escalona N. y González R. "Variante de recuperación de robot industrial RIAC-6." Trabajo de diploma, Facultad Ingeniería Eléctrica. 1998.

[KATRIB-83] Katrib, M. "Lenguajes de programación y técnicas de compilación." Pueblo y Educación, Cuba 1983.

[MSDN–98] Microsoft Developer NetWork. CD ROM abril-98. Microsoft cooperation.

[NIUBÓ–97] Niubó José: "Diseño de accionamiento para motor de corriente directa." Trabajo de diploma UCLV, Facultad Ingeniería Eléctrica. 1997.

[USÁTEGUI–90] Usátegui Angulo, J.M R. Avilés González. "Curso de Robótica." Ediciones Revolucionarias. La Habana 1990. 430p.

Comandos más importantes del editor de código

Ctrl+Ins: Pone una copia del texto seleccionado en el portapapeles y deja el texto original intacto.

Shift+Del: Elimina el texto seleccionado de su posición actual y lo pone en el portapapeles.

Shift+Ins: Inserta el contenido del portapapeles en la posición del cursor.

End: Mueve el cursor al final de la línea.

Home: Mueve el cursor al comienzo de la línea.

Enter: Inserta un cambio de línea.

Ins: Conmuta el modo de inserción activo / desactivo.

Del: Elimina el carácter a la derecha del cursor.

Backspace: Elimina el carácter a la izquierda del cursor.

Tab: Inserta una tabulación.

Space: Inserta un espacio en blanco.

Left Arrow: Mueve el cursor una columna a la izquierda.

Right Arrow: Mueve el cursor una columna a la derecha.

Up Arrow: Mueve el cursor una línea hacia arriba.

Down Arrow: Mueve el cursor una línea hacia abajo.

Page Up: Mueve el cursor una página hacia arriba.

Page Down: Mueve el cursor una página hacia abajo.

Ctrl+Left Arrow: Mueve el cursor una palabra a la izquierda.

Ctrl+Right Arrow: Mueve el cursor una palabra a la derecha.

Ctrl+Home: Mueve el cursor a la parte superior de la pantalla.

Ctrl+End: Mueve el cursor a la parte inferior de la pantalla.

Ctrl+Pgdn: Mueve el cursor al final del documento.

Ctrl+PgUp: Mueve el cursor al comienzo del documento.

Ctrl+Backspace: Elimina una palabra a la izquierda.

Ctrl+Del: Elimina el bloque seleccionado.

Ctrl+Tab: Cambia a la próxima página.

Shift+Ctrl+Tab: Cambia a la página anterior.

Alt+Backspace: Deshace la última operación de edición.

Alt+Shift+Backspace: Rehace la ultima operación deshecha.

Shift+Right Arrow: Selecciona el carácter a la derecha del cursor.

Shift+Left Arrow: Selecciona el carácter a la izquierda del cursor.

Shift+Up Arrow: Selecciona desde la posición actual del cursor hasta una línea más arriba.

Shift+Down Arrow: Selecciona desde la posición actual del cursor hasta una línea más abajo.

Shift+PgUp: Selecciona desde la posición actual hasta una pantalla más arriba.

Shift+PgDn: Selecciona desde la posición actual hasta una pantalla más abajo.

Shift+End: Selecciona desde la posición actual hasta el fin de la línea.

Shift+Home: Selecciona desde la posición actual hasta el comienzo de la línea.

F1: Muestra la ventana de ayuda.

F2: File | Save

F3: File | Open

F4: Run | Run to Cursor

F5: Restaura / Maximiza el tamaño de la ventana.

F6: Muestra la próxima página.

F7: Run | Trace Into

F8: Run | Step Over

F9: Run | Run

Alt+F3: File | Close

Alt+X: File | Exit

Ctrl+F2: Run | Program Reset

Ctrl+F8: Poner / Quitar un punto de ruptura.

Ctrl+F9: Project | Compile

Ctrl+Shift+Dígito: Pone un marcador (con numero Dígito) en la posición actual.

Ctrl+Dígito: Mueve el cursor a la posición del marcador con número Dígito.

Descripción de los comandos del menú

File: Permite abrir, cerrar, salvar e imprimir proyectos y archivos nuevos o existentes, así como agregar nuevos módulos al proyecto abierto.

New Module: Crea y adiciona un nuevo módulo al proyecto actual.

New Application: Crea un nuevo proyecto y su archivo .GAF asociado. Si un proyecto está abierto cuando usted escoge File |New Application, Gospel le propone salvar cualquier cambio al proyecto, cierra el proyecto actual, y crea uno nuevo.

Open Project: Permite abrir un proyecto existente. Si hay algún proyecto abierto se le propone salvar sus cambios y el proyecto actual es cerrado antes de abrir el nuevo.

En adición a los comandos descritos anteriormente el menú File contiene los comandos usuales **Open, Reopen, Save, Save As, Save All, Close, Close All, Print, Exit.**

Edit: Permite cortar, copiar y pegar texto, así como deshacer y rehacer operaciones en el editor de programas. Contiene los comandos usuales **Undo, Redo, Cut, Copy, Paste.**

Search: Permite localizar y remplazar texto en el editor de programas.

Go to Line Number: Mueve el cursor hacia el número de línea que se especifique.

El menú Search contiene, además, los comandos usuales **Find, Replace, Search Again.**

View: Permite mostrar u ocultar diferentes elementos del ambiente de Gospel.

Project Manager: Muestra al Project Manager, el cual lista todos los módulos presente en el proyecto actual. Usted puede usarlo para navegar entre los archivos del proyecto y, además, adicionar, eliminar, salvar o copiar un archivo al proyecto actual.

- Symbols:** Este comando del menú se usa para explorar visualmente, los distintos módulos y los símbolos que en ellos se definen: constantes, variables, tipos de datos, procedimientos y funciones.
- Project:** Permite compilar la aplicación.
- Add to Project:** Adiciona un archivo al proyecto.
 - Remove from Project:** Elimina un archivo del proyecto.
 - Syntax Check:** Hace el chequeo sintáctico y semántico de la aplicación pero no genera el código interno.
 - Compile:** Compila toda la aplicación.
 - Directories:** Muestra la caja de diálogo Directories, donde se especifica la localización de los archivos que se necesitan para la compilación.
- Run:** Los comandos de este menú permiten correr y depurar un programa.
- Run:** Compila y ejecuta la aplicación.
 - Step Over:** Ejecuta un programa de línea en línea, pasando por encima de los procedimientos y ejecutándolos como una sola unidad.
 - Trace Into:** Ejecuta un programa de línea en línea, entrando en los procedimientos y siguiendo la ejecución de cada línea.
 - Run to Cursor:** Corre el programa cargado hasta la posición del cursor en la ventana de editor de código.
- Pause:** Pausa temporalmente la corrida de un programa en ejecución.
- Reset:** Finaliza la ejecución del programa actual y lo libera de memoria.
- Options:** Permite establecer distintas opciones del ambiente.
- Font:** Muestra la caja de diálogo Fonts que permite seleccionar el estilo de letra que se desea usar en la edición.
 - Preferences:** Muestra una caja de diálogos para establecer las preferencias en la configuración del ambiente.
- Tools:** Permite configurar y ejecutar programas de apoyo.
- Configure Tools:** Muestra una caja de diálogo para adicionar, eliminar y editar programas en el menú Tools.

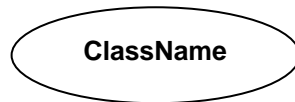
Introducción a la notación utilizada

Los diagramas de clases e instancias incluidos en el presente trabajo han sido representados utilizando la notación desarrollada por el Grupo Merchise, CEI UCLV. Este apéndice introduce elementos básicos de esta notación, de forma que los diagramas puedan ser interpretados cabalmente.

Los diagramas se basan en la metodología que actualmente desarrolla el grupo como parte de sus proyectos de producción. La metodología enfoca tres elementos principales: objetos-metaobjetos, interfaces y relaciones.

No existen diagramas de clases e instancias por separado, sino que estos se integran en un solo plano.

Las clases del lenguaje se representan de la siguiente forma:








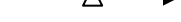


Las instancias del lenguaje se representan de las siguientes formas:



La representación con líneas dobles se utiliza a la hora de definir la instancia. Si no aparecen estas líneas significa que la instancia aparece en ese lugar del diseño, pero que es definida en otra parte.

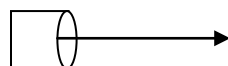
Las relaciones entre objetos se representan mediante flechas. Sobre la flecha se incluye un símbolo que denota el tipo de la relación. Se reconocen los siguientes tipos:

Nombre	Relaciona	Representación	Relación Dual
Uso	objeto-objeto		Uso
Contiene-a	objeto-objeto		Contenido-por
Se-instancia-en	metaobjeto-objeto		Se-clasifica-en
Se-especiliza-en	metaobjeto- metaobjeto		Hereda-de
Contenido-por	objeto-objeto		Contiene-a
Se-clasifica-en	objeto-metaobjeto		Se-instancia-en
Hereda-de	metaobjeto-metaobjeto		Se-especiliza-en
Implementa-a	objeto-interfaz		-

El término “objeto” se le aplica a cualquier identidad que no sea una relación. “Metaobjeto” es un objeto que actúa como “clase” o “género” de otro objeto, sea lo mismo una instancia o una clase del lenguaje.

La inclusión de una clase del lenguaje en el diagrama puede desecharse si solo tiene una relación “Se-instancia-en”.

Además, la representación varía en dependencia de la cardinalidad de la relación. Cuando se trata de un número fijo, este se sitúa encima de la flecha. En caso de ser uno, se ignora. Si se trata de un número variable se utiliza el siguiente símbolo:



Si el número se encuentra acotado en cierto rango, este rango se especifica al final de la flecha. Si el límite inferior en el rango es cero, entonces se ignora. El tipo de la relación se especifica en este caso de igual manera; o sea, sobre la flecha.