Universidad Autónoma de Querétaro
Facultad de Ingeniería
Licenciatura en
Matemáticas Aplicadas

MÓDULO DE INTERFAZ DE USUARIO PARA SISTEMAS DE CONTROL NUMÉRICO POR COMPUTADORA IMPLEMENTADO EN JAVA

TESIS

Que como parte de los requisitos para obtener el grado de

Licenciado en Matemáticas Aplicadas

Presenta:

Jared Piña Bárcenas

Dirigido por:

Dr. Roberto Augusto Gómez Loenzo

Centro Universitario
Querétaro, Qro.
Octubre de 2012
México

Universidad Autónoma de Querétaro

Facultad de Ingeniería

Licenciatura en Matemáticas Aplicadas

MÓDULO DE INTERFAZ DE USUARIO PARA SISTEMAS DE CONTROL NUMÉRICO POR COMPUTADORA IMPLEMENTADO EN JAVA

TESIS

Que como parte de los requisitos para obtener el grado de

Licenciado en Matemáticas Aplicadas

Presenta:

Jared Piña Bárcenas

Dirigido por:

Dr. Roberto Augusto Gómez Loenzo

Sinodales:

Dr. Roberto Augusto Gómez Loenzo	
Presidente	Firma
M. en I.S.D. José Luis Gonzáles Pérez	
Secretario	Firma
M. en C. Miguel Ángel Martinez Prado	
Vocal	Firma
M.D.M. Benjamín Zúñiga Becerra	
Suplente	Firma

Centro Universitario
Querétaro, Qro.
Octubre de 2012
México

RESUMEN

Las máquinas de control númerico son fundamentales para el desarrollo y competencia internacional de las empresas. El software es una pieza clave en el desempeño y resultados que se obtienen del uso de estas máquinas. En estos procesos es muy importante que exista una comunicación óptima entre la máquina y el usuario. Esto se logra por medio de una interfaz gráfica de usuario (GUI), la cual tiene como finalidad conseguir una comunicación visual efectiva con el usuario. Es más, la interfaz gráfica juega un papel muy importante para la aceptación por parte de los usuarios de cierto software.

En el presente trabajo se muestra el desarrollo de una interfaz de usuario para sistemas de control numérico por computadora (CNC) a dos pantallas, una normal y la segunda táctil (touchscreen). La interfaz es un módulo implementado en Java, que aprovecha las propiedades de este lenguaje de programación, como son la independencia de la plataforma y su capacidad de convivencia con código nativo. Esta convivencia entre la interfaz gráfica y el resto de los módulos desarrollados en C++ se realiza con un protocolo de comunicaciones estándar basado en interfaces (clases base abstractas de C++).

Palabras clave: GUI, CNC, Java, módulo, pantalla táctil, C++.

A mi madre
—por todo su apoyo.

AGRADECIMIENTOS

A mi Madre, sin ella esto no es posible.
A mi familia, por su cariño.
A todos mis maestros, por todos los conocimientos brindados.
A mis amigos, por todas las vivencias compartidas.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

I. INTRODUCCIÓN

Con la incorporación de tecnología en los diferentes tipos de máquinas-herramienta (CNC), se puede lograr una mayor eficiencia en los procesos de producción, reflejada en mayor calidad y menores tiempos de fabricación y costos. Estas máquinas son fundamentales para el desarrollo y competencia internacional de las empresas y por lo tanto una pieza importante para el desarrollo de la economía nacional. En México existe una fuerte dependencia a la importación de este tipo de máquinas. Si se continúa el desarrollo de estos equipos, se podrá evitar la fuga de este capital y se promoverá la sustentabilidad de la base manufacturera del país.

Dentro del desarrollo de máquinas herramienta, el software juega un papel muy importante, pues este es el encargado de que la máquina realice las tareas necesarias. Al manejar una máquina CNC se genera demasiada información sobre el estado de la máquina, posición, tareas, y demás. Es necesario tener un medio de controlar y organizar estos datos y que ayude a la gente a comprender información compleja. Esto se logra por medio de una interfaz gráfica de usuario, la cual tiene como finalidad conseguir una comunicación visual efectiva con el usuario. La interfaz gráfica juega un papel muy importante para la aceptación por parte de los usuarios de cierto software.

En el artículo del Equipo Editorial de Metalmecánica Internacional (2009) se menciona que las grandes potencias mundiales como Japón, Estados Unidos y Alemania aumentaron su producción y exportación de máquinas herramienta a nuestro país. Otras comienzan a crecer siendo el caso de China, Corea del Sur y Taiwán. Actualmente, México vive en un rezago, pues no existe una producción nacional de máquinas y herramientas suficiente para satisfacer la demanda y como si esto fuera poco, es muy difícil acceder a créditos enfocados a la adquisición de nueva maquinaria para mejorar la producción en las empresas. Por ejemplo, en el caso de Estados Unidos, aproximadamente cada diez años se renueva la tecnología de las empresas y ocurre que muchas compañías mexicanas compran esa tecnología usada, pero en algunos casos siguen trabajando con máquinas herramienta de la década de los cuarenta, las cuales han sido renovadas de forma muy esporádica y con equipos de bajo costo.

De acuerdo a la información estadistica proporcionada por la subsecretaría Industria y Comercio (2012), en los últimos cuatro años se han importado un total de 738 máquinas fresadoras de consola de control numérico al país para lo cual hemos tenido como principales proveedores Estados Unidos, Taiwan, Japón, Corea, Alemania y España, por nombrar algunos. Estas adquisiciones han representado una fuga de capital hacia el extranjero que asciende a 93,529,051 dólares. Por otro lado, las exportaciones de estas máquinas solo asciende a 14 máquinas en los correspondientes cuatro años siendo Estados Unidos y Canadá los mayores clientes. Estas ventas han significado una ganancia de tan solo 686,852 dólares. Como otro ejemplo, se han importado 11,756 centros de mecanizado con un costo de 1,043,733,329 dólares, contra los 199 centros exportados con una ganancia de 22,011,092 dólares. La dife-

rencia es significativa, y lamentablemente como consecuencia México pierde al año una gran cantidad de capital en la importación de estas máquinas.

En el equipo industrial mexicano viene ganando terreno el sistema de control numérico por computadora (CNC) frente a los sistemas mecánicos convencionales. En la industria de autopartes, donde México se encuentra entre los países latinoamericanos con mayor inversión extranjera, las máquinas de fresado, torneado y roscado son indispensables para los procesos de producción.

I.1. Antecedentes

Una GUI es un método para facilitar la interacción del usuario con la computadora por medio del uso de imágenes, tales como ventanas, iconos, botones, etc., además de texto.

Aunque actualmente el uso de las GUIs es muy común, no siempre ha sido tan sencilla la forma en que las personas han interactuado con las computadoras. En el año de 1837 el matemático Charles Babbage (1793–1871) quiso aplicar el concepto de las tarjetas perforadas del telar de Jackard inventado en 1801, por medio de su máquina analítica para ingresar datos y programas. El lenguaje de programación que se propuso para la máquina analítica había sido muy parecido a los actuales lenguajes ensambladores, para lo cual se utilizaron tres tipos de tarjetas perforadas:

- Para operaciones aritméticas.
- Para constantes numéricas.
- Para operaciones de almacenamiento y recuperación de datos de memoria, y transferencia de datos entre la unidad aritmética y la memoria.

Esta máquina disponía de tres lectores diferentes, uno para cada tipo de tarjeta, sin embargo, desafortunadamente no fue construida por razones de índole financiero y político.

En 1843 Lady Ada Augusta Lovelace sugirió la idea de que las tarjetas perforadas pudieran adaptarse de manera que propiciaran que la máquina analítica de Babbage repitiera ciertas operaciones. Debido a esta sugerencia algunas personas consideran a Lady Lovelace la primera programadora.

Durante las décadas de 1960 y 1970, se hizo común el uso de tarjetas perforadas. Dichas tarjetas funcionan de la siguiente forma: cada tarjeta es una cartulina con determinadas posiciones que pueden o no estar perforadas, tal como si fuese código binario. A cada número o letra le corresponde una columna. Los números tienen una sola perforación en la columna, mientras que las letras tienen dos perforaciones. También se pueden representar con perforaciones algunos símbolos especiales, tales como: coma (`,`), punto (`.`), suma (+), guión (−), asterisco (⋆), diagonal (/), entre otros. Estos símbolos pueden tener dos o tres perforaciones por columna. Sin embargo no todos usaban los mismos símbolos y claves.

Para introducir la información a una computadora por medio de tarjetas perforadas, éstas se alimentaban en un lector de tarjetas el cual contaba con unas escobillas o celdas fotoeléctricas que detectaban la presencia de agujeros, las cuales eran capaces de generar una serie de impulsos eléctricos que los circuitos de la unidad de control sabían interpretar y enviaban a la memoria o unidad aritmética. Un lector de tarjetas típico podía "leer"de 100 a 2000 tarjetas por minuto.

Las tarjetas perforadas también funcionaron como mecanismo de salida, solo que en este caso actuaban de manera inversa. Es decir, en lugar de tener un lector de tarjetas se usaba una mecanismo de salida que las perforaba. Naturalmente se utilizaban las mismas claves, de manera que una tarjeta que se obtuviera de una perforadora podía ser leída subsecuentemente por una lectora de tarjetas sin ningún procesamiento adicional.

Posteriormente, durante la década de 1950 surgió la línea de órdenes, mejor conocida como intérprete de comandos o *shell*. Una línea de órdenes es un método para interactuar con la computadora por medio de una terminal de texto. En este caso el usuario introduce órdenes por medio de lineas de texto desde un teclado, y también recibe una salida en forma de texto.

La línea de órdenes supuso un avance sobre el uso de tarjetas perforadas puesto que la interacción y la retroalimentación con la computadora se agilizó de manera significativa.

La forma más simple de una línea de órdenes muestra un símbolo de petición de órdenes, en el cual el usuario teclea una orden y la finaliza normalmente con la tecla *Intro*, la computadora ejecuta esta orden, y finalmente proporciona una salida de texto. Las órdenes de un intérprete de comandos generalmente se encuentran en alguna de las siguientes formas:

```
[haz_algo] [de_esta_manera] [a_estos_archivos]

[haz_algo] < [archivo_de_entrada] > [archivo_de_salida]
```

en donde el comando u orden `haz_algo` indica la acción que se va a realizar, `de_esta_manera` es información adicional para explicar la manera en que se va a llevar a cabo la acción y `a_estos_archivos` sugiere uno o varios objetos sobre los cuales debe actuar la orden.

El carácter '>' de la segunda forma es un carácter de redirección, que le dice a la computadora que no envíe la salida de la orden a la pantalla, sino al archivo nombrado a la derecha de '>', y el carácter '<' es un indicador que precede al nombre del archivo que contiene los datos de entrada del programa.

Así mismo, tiempo después durante la década de 1960, Douglas Engelbart, mejor conocido por haber inventado el ratón de computadora, fue el pionero de la interacción humana con los ordenadores, desarrollando la primera GUI en los laboratorios de Xerox, Palo Alto (California, EE.UU.), misma que fue introducida posteriormente en las computadoras Apple de Macintosh en 1984 y, hasta 1993, en la primera versión de Windows 3.0.

Desde ese momento las interfaces gráficas se han vuelto muy populares, sobre todo porque permiten que una persona interactúe de una manera mucho más sencilla con las computadoras, ya que no requiere del conocimiento del código binario como en el caso de las tarjetas perforadas, para lo cual tampoco es necesario aprender diversas órdenes o comandos para realizar las actividades como sucede con las líneas de órdenes.

Además, existen algunas interfaces gráficas que han sido diseñadas con requerimientos específicos para cierto tipo de mercados, las cuales se conocen como "GUIs de uso específico". Un ejemplo de este tipo de interfaz viene a ser el llamado *touchscreen*, la cual es una pantalla que al tocarla efectúa comandos que normalmente realizaría el ratón.

I.2. Justificación

Un programa de propósito general, destinado al control de máquinas herramienta, es EMC (*Enhanced Machine Controller*) basado en un kernel Linux en tiempo real, cuyo código es de libre distribución[1] y desarrollado por Linux CNC (2012).

Este programa fue desarrollado por el NIST (*National Institute of Standards and Technology*), el cual anteriormente formaba parte del *Commerce Department of the United States government*. Primeramente se encontraban interesados en realizar un "paquete de control de movimiento" como plataforma de prueba para conceptos y normas, para lo cual la aportación de *General Motors* permitió una adaptación inicial de EMC, usando tarjetas de control PMAC y corriendo bajo la versión de "tiempo real" de Windows NT, para controlar una fresadora grande. Posteriormente se optó por utilizar la extensión de "tiempo real" de Linux, en lugar del costoso y temperamental sistema Windows NT, para lo cual se creó un disco con el sistema completo y funcional para lograr ser instalado. También se decidió cambiar la costosa tarjeta de control por una tarjeta *ServoToGo* y aprovechar el poder de procesamiento del ordenador. Así mismo, se añadió software a la interfaz de usuario para planificar trayectorias, control de bucles PID[2] y un intérprete RS274. Después de una reestructuración del código para hacer un entorno de desarrollo más amigable, se dio lugar al nacimiento del EMC2.

El EMC2 proporciona un sistema de control de movimientos para un máximo de nueve ejes, un intérprete del lenguaje estándar RS274 (de códigos G), incluyendo diversas interfaces gráficas de usuario entre las que se encuentran simulaciones 3D con OpenGL (*Open Graphics Library*) [3], una estructura sencilla para la configuración de cada máquina y soporte de sistemas no cartesianos. Este software se conforma por cuatro partes fundamentales, una es encargada de controlar el movimiento de los motores (EMCMOT), otra controla las entradas y salidas digitales (EMCIO), una más es un programa que ejecuta las tareas y coordina así los dos anteriores (EMCTASK) y la última es un conjunto de interfaces gráficas de usuario que se describen a continuación.

MINI Interfaz gráfica diseñada para trabajar a pantalla completa, originalmente desarrollada para las maquinas CNC Sherline.

AXIS Interfaz gráfica *front-end*[4] que cuenta con vista previa en vivo y representación gráfica del código. Está escrito en Python y utiliza Tk y OpenGL.

Touchy Interfaz gráfica para uso en el panel de control de la máquina y no requiere raton ni teclado. Esta destinada a ser usada con una pantalla táctil en combinación con una perilla y un panel de operación manual.

KeyStick Interfaz gráfica simple basada en texto.

[1] Licencia GNU GPL.

[2] Controlador Proporcional Integral Derivativo.

[3] Especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D

[4] Software que interactua con el usuario, responsable de recolectar los datos de entrada, que pueden ser de muchas y variadas formas.

Por otro lado, el desarrollo de controladores numéricos en México se estableció con la aparición del CHROM-1, siendo éste el primer control numérico creado(Herrera Ruiz and Molina (1989)). El CHROM-1 incorporaba diversas características comunes presentes en sistema de control numérico moderno, tales como interpolación lineal en 3 ejes, interpolación circular en los planos XY, YZ y XZ y diversos ciclos enlatados. Su interfaz gráfica con el usuario era estándar para la década de los 80s, ya que en ese momento predominaban las líneas de comandos e interfaces gráficas basadas en caracteres. El desarrollo del CHROM-1 continuó en los proyectos CHROMA-1 y CHROMA-2, en los cuales el Dr. Pedro Daniel Alaniz Lumbreras realizó mejoras paulatinas a diversos subsistemas del mismo. Durante el desarrollo del CHROMA-1 se extendieron las capacidades del sistema para operar con motores de corriente directa con retroalimentación y se ampliaron las capacidades del intérprete para operar un administrador automático de herramientas.

Al mismo tiempo que se desarrollaba el proyecto CHROMA-1 en el Laboratorio de Mecatrónica, se implementaba en el Laboratorio de Biotrónica un Sistema de Control Climático Inteligente (SCCI) para Invernaderos, bajo la iniciativa del M. en C. Juan José García Escalante y el Dr. Rodrigo Castañeda Miranda. En su primera versión presentada en enero de 2000 dicho sistema permitía administrar, por medio de riego programado, hasta 32 riegos por dos secciones. En julio de ese mismo año el sistema fue extendido para operar el sistema de calefacción permitiendo 32 programaciones para el mismo. Un año después, en Julio de 2001 se integró el control de ventanas, permitiendo ocho programaciones para las mismas. A finales del año 2001 se iniciaron las primeras pruebas de la primera GUI experimental para sistemas TUNA (Tecnología Universitaria en Automatización), empleándose para la misma 256 colores y una resolución de 800x600 pixeles, la cual fue integrada en la versión del SCCI de enero de 2002. El sistema gráfico continuó siendo mejorado para las versiones TUNA SCCI 5.0 de enero de 2003 y TUNA SCCI 5.2 de diciembre de 2003. Estas últimas versiones del sistema gráfico se caracterizaron por tener un grado de independencia relativo del resto del código de las aplicaciones mencionadas. Dicho sistema gráfico ofrecía, en consecuencia, la posibilidad de ser portado a otros sistemas TUNA.

Fue en enero de 2004 que el Dr. Pedro Daniel Alaniz Lumbreras realizó los estudios y la planeación necesaria para proveer al sistema CHROMA-2 de una GUI parecida a la que ya formaba parte del sistema TUNA SCCI 5.2, con ligeras modificaciones apropiadas para sistemas de control numérico por computadora para máquinas herramienta. Dicha implementación, sin embargo, integra de manera no modular la GUI con el resto de la aplicación. En virtud de los requerimientos establecidos por el nuevo proyecto, el diseño propuesto ahora es modular basado en una biblioteca que emplea recursos binarios en memoria, cargados a partir de un archivo de recursos en disco. Esta es una de las razones de ser del presente proyecto, junto con la intención de crear una biblioteca para interfaces gráficas de usuario que esté disponible y sea apropiada para una amplia gama de proyectos desarrollados en nuestra institución.

Una de las mejores formas para reducir la complejidad de un problema, sobre todo cuando se trata de productos integrados por varios componentes, consiste en dividir o particionar estos sistemas en componentes de menor tamaño, para que se puedan implementar de forma mucho más sencilla. En términos de software a este proceso se le llama **modularización**, y a los elementos que resultan de este proceso son llamados **módulos**, entendiéndose por módulo cada unidad de software que contiene funciones e interfaces bien definidas para

interactuar con otros elementos del programa. Cada módulo debe de ser lo suficientemente simple como para que nos permita:

- Comprender su propósito y estructura.
- Verificar (que el resultado corresponde al diseño original) y validar (verificar que el diseño resuelva el problema).
- Apreciar su interacción con otros módulos.
- Valorar su efecto sobre la estructura general del software y su operación.

I.3. Hipótesis y objetivos

El principal objetivo de este trabajo es mostrar que es posible implementar una GUI con Java para un control de máquinas CNC en C++ siempre y cuando exista un protocolo de comunicaciones estándar entre los diferentes módulos.

En particular, para alcanzar este objetivo general, se establecen los siguientes objetivos particulares:

1. Establecer un protocolo de comunicaciones estándar entre una GUI y el resto del sistema de CNC.
2. Implementar una GUI para sistemas de CNC en el lenguaje Java.
3. Crear una interfaz entre el código Java y el protocolo de comunicaciones definido en C++.
4. Verificar que la integración con el resto de los módulos del sistema es la apropiada, generando entradas al sistema para detectar defectos.

A partir de estos objetivos, se desea poner a prueba la hipótesis de este trabajo, la cual consiste en comprobar que:

> Si existe un protocolo de comunicaciones estándar entre el módulo de interfaz gráfica de usuario y el resto de los diferentes módulos, entonces, es posible realizar el desarrollo del módulo de interfaz gráfica de usuario con Java para un sistema de CNC en C++.

I.4. Descripción del problema

Un software monolítico es difícil de entender, debido al gran numero de variables, funciones y a la complejidad global del mismo. El dividir en subprogramas ayuda a resolver de forma sencilla un problema más general.

Una GUI creada como un módulo cuenta con ciertas ventajas, ya que el tamaño del código es relativamente más pequeño, permitiendo a su vez que la localización y corrección de errores sea más sencilla y se logre en menor tiempo. También se pueden realizar cambios

en la estructura del programa por rediseño de algoritmos sin afectar otros módulos al trabajar como "caja negra" [5]. En este caso la independencia modular permite trabajar de forma paralela con otros módulos, debido a que el programa puede ser compilado sin la necesidad de tener el código completo. Además, una interfaz gráfica de usuario desarrollada de esta forma, puede ser adaptada a otros programas como por ejemplo un simulador de un CNC.

[5]Elemento que es estudiado desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno.

II. REVISIÓN DE LA LITERATURA

En el presente capítulo se muestran, en dos secciones, trabajos relacionados con el tema de la presente tesis. En la primera sección se analizan las consideraciones que se deben tener al realizar una GUI. En la segunda sección se tratan algunas interfaces que se han realizado para máquinas de control numérico.

II.1. Consideraciones

Dentro de las consideraciones a tomar al realizar una GUI, hay que determinar primero sus funciones básicas. En este contexto, Molina Moreno (2003) dice que si se descompone un sistema en capas lógicas según su función, se puede ver a la interfaz de usuario como a la capa lógica encargada de dar soporte al diálogo con el usuario. Sus funciones, son esencialmente dos:

Entrada: adquirir las ordenes lógicas según su función, se puede ver a la interfaz de usuario como a la capa lógica encargada de dar soporte al dialogo con el usuario a través de diversos dispositivos de interacción.

Salida: presentar resultados, retroalimentación y cooperar para facilitar al usuario la realización de las tareas que pretende resolver el sistema.

Conforme las prestaciones de los equipos de cómputo han ido creciendo, se a dado paso de las líneas de comandos, a los menús contextuales, hasta llegar a las actuales interfaces gráficas de usuario. Esto cambió la forma de diseñar las interfaces de usuario. Por lo tanto, hay que determinar las partes que forman una interfaz de usuario moderna. Molina menciona que todos los entornos gráficos de usuario tienen elementos en común. Estas características comunes están encerradas bajo las siglas WIMP acuñadas en Palo Alto que son el acrónimo en inglés de ventana, icono, menú y puntero (*Window, Icon, Menu, Pointer*). A continuación se describe cada uno de estos elementos.

Window Ventanas y distribución del área de trabajo.

Icon Representación gráfica de un objeto manipulable.

Menu Selección de objetos o acciones.

Pointer Puntero para seleccionar objetos y/o acciones a través de dispositivos como: ratón, teclado, lápiz óptico, pantallas táctiles, guantes de datos, etc.

Otro punto importante a tomar en cuenta es la consistencia. Expósito (2006) dice que la consistencia en el diseño es el proceso mediante el cual se establece a la hora de estructurar

menús, comandos y elementos de navegación en la interfaz, un orden común y coherente. De este modo, el usuario sólo tiene que aprender una sola vez donde localizar las acciones en los menús, y aunque se produzca un cambio en la aplicación, sepa localizarlos sin problemas. La consistencia en el diseño de interfaces, es un elemento muy importante porque reduce la curva de aprendizaje del sistema por parte del usuario.

También se menciona que la interfaz ha entrado en un nuevo período y se ha transformado en superficie inteligente. Los procesos de inteligencia añadidos a la interfaz, han convertido a ésta en un autómata inteligente, capaz de tomar decisiones propias sobre su propia forma, en lo que respecta al modo de estructurar y organizar elementos en la misma interfaz. Por lo tanto, ya no podemos considerar la interfaz gráfica como un mero artefacto interactivo. La interfaz ha sido dotada de inteligencia artificial, muy rudimentaria y por lo tanto ha sido transformada en superficie inteligente capaz de ayudarnos a tomar decisiones.

Por último, Rivera Loaiza (2000) opina que las interfaces de usuario (IU) modernas deben ser escritas de adentro hacia fuera. En vez de organizar el código para que la aplicación tenga el control, la aplicación debe más bien estar dividida en muchas subrutinas que son llamadas cuando el usuario realiza alguna acción. Se requiere de una programación y modularización del software de la IU. Rivera continua señalando que desafortunadamente, la separación de estas dos partes (la IU y el resto del software) es muy difícil, prácticamente imposible, ya que los cambios en la IU requieren inevitablemente cambios en el resto del software. Incluso con la utilización de herramientas para la creación de interfaces de usuario el problema de modularidad se hace más difícil por la gran cantidad de funciones *call-back* [1]. Generalmente cada *widget* en la pantalla requiere que el programador escriba al menos un procedimiento de aplicación a ser llamado cuando el operador lo activa. Cada tipo de *widget* tendrá su propia secuencia de funciones *call-back*. Una puede contener miles de *widgets*, por lo que habrán al menos la misma cantidad de funciones *call-back*.

II.2. Opciones de interfaces CNC

Antes de contar con una interfaz gráfica de usuario, los equipos de computo y en especifico los controles numéricos ofrecían una interfaz de línea de comandos. Es propuesto por ISO TC 184/SC 1 (1981) una estandarización de comandos operativos y formatos de datos para máquinas de control numérico. La interfaz contaba con comandos para permitir al usuario elegir entre los distintos modos de operación del sistema. En el reporte se ilustran tres modos de operación con sus comandos de acceso correspondientes: modo editor, administrador de archivos y control de la máquina. También existe la posibilidad de agregar modos adicionales y definir sus comandos de acceso, para adaptarse a las necesidades de cada sistema de control numérico.

Este tipo de interfaces aun se siguen implementando, por ejemplo en prototipos como el realizado por Villarreal (2008) para el torneado en serie de metales. Explica que puesto que no se ha diseñado ningún módulo de control intermedio entre el PC y la máquina, el volumen de información a enviar a través del puerto de comunicación es relativamente grande y depende de la resolución escogida para los controladores de los motores de paso. Por lo tanto, para hacer uso exclusivo de todos los recursos del PC se implementó el software

[1] Call-back o devolución de llamada es una función “A” que se usa como argumento de otra función “B”. Cuando se llama a “B”, ésta ejecuta “A”. Para conseguirlo, usualmente lo que se pasa a “B” es el puntero a “A”.

de control bajo DOS, un sistema operativo monoproceso, y utilizando C++ como plataforma de desarrollo.

Sin embargo, Molina explica que ahora resulta muy sencillo crear interfaces gráficas de usuario con las llamadas herramientas RAD (*Rapid Application Development*). En ellas, el programador y/o diseñador de la interfaz de usuario ayudados por un IDE van construyendo la interfaz de usuario mediante el paradigma WYSIWYG[2]. Concluye que la experiencia del diseñador es crucial para obtener interfaces de alta calidad. Las interfaces diseñadas de este modo son dependientes de un lenguaje de programación o librería de controles dada, dificultando la portabilidad a otros entornos.

Un ejemplo de lo anterior es el trabajo de Gordon and Hillery (2005), quienes decidieron crear la interfaz basada en Windows con el entorno de programación *Borland C++ Builder 3*. Proponen una GUI que, en la medida de lo posible, proporcione las mismas funciones que se encuentran en un control de una máquina herramienta convencional. Se decidió incorporar las siguientes características en la interfaz:

1. Soporte para programas de código G y un editor interactivo.

2. Modo retorno a cero, avance constante y editor de datos.

3. Simulación gráfica del programa.

4. Posición de los ejes.

Al igual que el entorno de desarrollo Builder, Visual Basic es muy socorrido para crear interfaces de manera rápida y sencilla. Muriel Escobar and Giraldo Giraldo (2010) crea una GUI para un torno que cuenta con la posibilidad de introducir código y de simular la trayectoria de la herramienta antes o simultáneamente con el proceso de maquinado. Se eligió Visual Basic para su desarrollo, debido al fácil manejo gráfico y por que permite la comunicación con el entorno a través del puerto serial, afirma.

Por su parte, Ospina et al. (2005) diseño una interfaz gráfica para un torno bajo Visual Basic, para el maquinado manual y automático de las piezas, permite ejecutar, visualizar y operar los diferentes movimientos del sistema.

Las desventajas de los entornos anteriores, concernientes a la portabilidad, se pueden prescindir con sistemas de desarrollo multiplataforma, como es el caso siguiente. La capa de interfaz gráfica de usuario que propone Ji et al. (2008) es únicamente una interfaz gráfica usuario-maquina, la cual es responsable de aceptar las instrucciones del usuario, la manipulación y desplegado de información del maquinado. Ha sido desarrollada en Tcl/TK, el cual es un lenguaje interpretado y es más lento que los lenguajes compilados. La capa de GUI no es responsable de los datos en tiempo real o tiempo de procesamiento, y no está diseñado para tal fin. Sin embargo, esta metodología de diseño simplifica la jerarquía y desarrollo del sistema. Además, otorga una interfaz de usuario mas amigable con menos esfuerzo.

También existen propuestas que aprovechan las ventajas que brinda la red. Álvarez and Ferreira (2006) proponen una interfaz gráfica de usuario para teleoperación de un centro de torneado CNC. La GUI (cliente-servidor) es implementada usando tecnologia Web, especialmente HTML, JavaScript y Java. La GUI consiste de una serie de Applets y formularios

[2]Acrónimo de *What You See Is What You Get*, paradigma conocido como programación visual.

HTML, los cuales son cargados por el explorador. La pantalla de estado permite visualizar la posición, modo de operación y la configuración global del CNC. Es posible ejecutar 300 funciones asociadas al control del CNC, PMC y DNC.

III. METODOLOGÍA

III.1. Java

Java se inicio en 1991 y fue diseñado por un grupo de ingenieros de Sun Microsystems (*Stanford University Network*) en Santa Clara, California E.U., liderados por el desarrollador de software Patrick Naughton y el doctor en Ciencias de la Computación, James Gosling. En un inicio el proyecto se nombro *Green*, el cual comenzó a desarrollarse con el objetivo de crear un lenguaje de programación destinado a electrodomésticos (dispositivos electrónicos inteligentes, como televisores, vídeos, equipos de música, etc.), independiente de la plataforma y del sistema operativo pues los diferentes fabricantes pueden seleccionar distintas unidades centrales de proceso (CPU), por esto era importante que el lenguaje no estuviera asociado a una sola arquitectura(Horstmann and Cornell (2006a)).

Basaron su lenguaje en C++, pero a diferencia de éste, se diseñó desde sus orígenes como verdadero lenguaje orientado a objetos. Además, los requisitos de pequeñez, compacidad y neutralidad respecto a las plataformas llevaron a diseñar un lenguaje transportable que generaba el código intermedio de una máquina hipotética. (Éstas suelen llamarse máquinas virtuales, dando inicio a la máquina virtual de Java o JVM). Es la maquina virtual quien interpreta el código intermedio llamado *bytecode*, conformado por instrucciones muy optimizadas en códigos de bytes, convirtiéndolo a código particular de la CPU utilizada.La máquina virtual de Java es la que depende de la arquitectura de la computadora, hay una específica para cada dispositivo, ya sea un teléfono móvil, un microondas, un PC con sistema operativo Linux, Windows o MacOS(Luis and Matilde (2001)).

Es importante enfatizar que ninguna compañía de electrodomésticos se intereso en Java, en especifico, el mercado de la televisión interactiva no era maduro. En realidad su verdadero potencial no se vio, sino hasta el año de 1995, cuando se presentó el navegador HotJava el cual estaba programado totalmente en Java y además se podía ejecutar código incluido en las páginas web (lo que hoy conocemos como applets[1]), lo cual dio inicio al interés en el proyecto por parte de los directores de SunLabs, mostrándose el potencial que tiene tanto como lenguaje de internet como lenguaje de propósito general(O'Connell (1995)).

Los programadores de Java describieron su lenguaje en el artículo "*The Java Language: An Overview*" [2]con las siguientes características: "Java es sencillo, orientado a objetos, distribuido, robusto, seguro, neutro respecto a arquitectura, adaptable, interpretado, de alto rendimiento, multihilo y dinámico". A continuación más a fondo que refieren de Java estas características.

[1]Componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo un navegador web.

[2]El articulo se encuentra en http://java.sun.com/docs/overviews/java/java-overview-1.html

Sencillo. A pesar de que Java esta basado en C++(que ha sido creado añadiendo extensiones orientadas a objetos a un lenguaje de programación clásico como es C), Java no arrastra ciertas características difíciles de comprender y que por esta razón en algunos casos conllevan una mala aplicación en la practica. Son el caso de la aritmética de apuntadores, ficheros de encabezado, estructuras, uniones, sobrecarga de operadores, clases base virtuales, entre otros. Todo en Java es una clase, es solo necesario entender la programación orientada a objetos. Además, la recolección de basura (que es la tarea de liberar memoria), en Java es automática mientras que en C++ el desarrollador tiene que programarla.

Orientado a objetos. La programación orientada a objetos modela el mundo real, esto es, cualquier ente de nuestro entorno se puede ver como un objeto que pertenece a una clase. Se dice que un lenguaje de programación es orientado a objetos cuando se crean, manipulan y construyen objetos. Los objetos deben tener propiedades (datos que los definen, variables) y un comportamiento (métodos que los representan). Entre las características más importantes que podemos encontrar de la orientación a objetos se encuentran:

- Abstracción. Determinar las características esenciales de un conjunto.
- Encapsulamiento. El acceso a la información de la clase es controlado y tiene un mejor orden.
- Modularidad. La capacidad de dividir en partes mas pequeñas una aplicación ,llamadas módulos, que son independientes e interactúan entre si.
- Polimorfismo. Permite tratar de forma genérica objetos de distintas clases, ahorrando así código y proporcionando simplicidad.
- Herencia. La capacidad de crear clases derivadas de otras manteniendo sus características y funcionalidad con la posibilidad de adaptarlas o mejorarlas a lo requerido.

Distribuido. Java cuenta con una amplia biblioteca de rutinas para fácilmente hacer frente a los protocolos TCP/IP, como son HTTP y FTP. Esto hace que la creación de conexiones de red sea mucho más fácil que en C o C++. Las aplicaciones en Java pueden acceder a la información de la red por medio de las direcciones URL, con la misma facilidad que si se accediera a un sistema de archivos local.

Robusto. Cuando se programa se esta sujeto a la posibilidad de cometer errores, al compilar (traducir el lenguaje de programación al lenguaje maquina), algunos de estos se pueden encontrar y corregir. La cantidad que se pueden controlar de esta forma es mínima en la mayoría de los lenguajes, ocasionando que estos sean detectados hasta la hora de ejecución de la aplicación. Java en cambio da un gran giro a esta situación, pues su compilador detecta un gran numero de errores. Además no se pueden cometer errores de asignación de memoria, pues como se mencionaba anteriormente, Java no soporta punteros (variables de dirección de memoria).

Seguro. Los programadores de Sun se han esforzado en hacer de Java un lenguaje de internet seguro. Por ejemplo, algunos gusanos y virus basan su ataque desbordando la pila de ejecución[3], lo cual han logrado que sea una tarea imposible de realizar por medio de una aplicación de Java. También resulta imposible que una aplicación errante corrompa la memoria. Además estas aplicaciones no pueden leer o escribir ficheros sin permiso y existen clases con firmas digitales, para tener cierta seguridad al saber quien fue el desarrollador de dicha aplicación y así determinar que privilegios se le pueden otorgar.

Neutro respecto a arquitectura. Arquitectura neutra quiere decir que es independiente de la plataforma. Una aplicación desarrollada en Java tiene la ventaja de ser compilada una vez y ser ejecutada en cualquier sistema operativo que cuente con el entorno de ejecución de Java. Aunque esta característica puede no ser del todo buena, pues al ser una maquina virtual la que se encargue de la interpretación de *bytecodes* es más lenta la aplicación. Sin embargo hoy en día esto es casi imperceptible por el gran avance que existe en cuanto al rendimiento del hardware.

Portable. En lenguajes como C/C++ existen dos grandes problemas: el tamaño de los datos primitivos y las interfaces. Los datos primitivos son básicos para realizar aplicaciones, en C existen 5 tipos diferentes mientras que en Java son 8, los mas comunes son los enteros, flotantes y caracteres. El tamaño de un entero de tipo `int` puede variar dependiendo del compilador y del sistema operativo, por ejemplo, en Windows 3.1 tiene un tamaño de 16 bits pero en Windows NT es de 32 bits. En Java un `int` es de 32 bits siempre, de igual forma cada uno de los restantes siete tipos de datos primitivos tienen un tamaño definido e independiente de la plataforma.

Hacer una interfaz de usuario para una aplicación y que se vea bien en cualquier sistema operativo es muy difícil. Sin embargo, en Java esto no es así. Se puede crear una de forma independiente a la interfaz de usuario que se tiene como base. Esto es una gran ventaja por que la interfaz de una aplicación desarrollada en Java se vera bien sin importar el sistema operativo y por lo tanto no sera necesario realizar adaptaciones.

Interpretado. En los lenguajes de alto rendimiento no interpretados el compilador traduce el programa a código maquina de forma directa. Este código solo puede ser ejecutado en la maquina nativa. Si se quisiera ejecutar en algún otro entorno el programa debería ser compilado en este. Como ya había sido mencionado, la maquina virtual de Java se encarga de la interpretación de un código neutro, resultado de la compilación del programa, para ser ejecutado en cualquier sistema.

Alto rendimiento. Antes mencionábamos que para compilar nuestro programa hacemos uso de un interprete conocido como maquina virtual, este proceso es lento en comparación con la compilación directa en la maquina nativa. Esta característica carece de un alto rendimiento. Sin embargo, existe la compilación en tiempo de ejecución (JIT, *just-in-time*) la cual optimiza la compilación a *bytecodes*. Este opción de compilación permite traducir el *bytecode* a código máquina nativo en tiempo de ejecución, es decir, cuando la aplicación esta siendo usada y almacenándolo para ser usado posteriormente si es necesario.

[3]Estructura dinámica de datos encargada de manejar las subrutinas.

Multihilo. Multihilo es la capacidad de realizar múltiples tareas al mismo tiempo, por ejemplo, una aplicación servidor en Java puede atender a varios clientes al mismo tiempo. Java es multitarea sin la necesidad de acceder a procedimientos especiales del sistema operativo nativo.

Dinámico. Java es un lenguaje dinámico por que permite actualizar clases en tiempo de ejecución, C++ carece de esta característica. Un ejemplo de esto son los programas en navegadores de internet.

Estas son las características con las que cuenta Java y las que lo hacen una excelente opción como lenguaje de propósito general.

Cuando se descarga Java desde su pagina oficial de descargas[4], se observa que existen diferentes ediciones. Cada una esta enfocada a un ámbito diferente. J2SE (*Java Standard Edition*) es la edición para las computadoras personales y es la necesaria para desarrollar aplicaciones. J2EE (*Java Entreprise Edition*) es la edición destinada para las empresas. Esta se puede ver como una extensión de J2SE, pues cuenta con todas las clases de esta, pero agrega otras enfocadas especialmente al desarrollo de servicios web, servicios de nombres, persistencia de objetos, XML, autenticación, API para la gestión de transacciones, etc. J2ME (Java Micro Edition) es la edición para realizar aplicaciones para dispositivos móviles. Los dispositivos móviles se caracterizan por tener pocos recursos y capacidades gráficas reducidas. Es una versión reducida del J2SE mas algunas clases especiales.

Al descargar J2SE se encuentran dos opciones: JDK y JRE. El JDK (*Java Development Kit*) contiene todas las herramientas necesarias para desarrollar aplicaciones Java, como son el compilador Javac, el visor de applets Appletviewer o el interprete de Java. En cambio, el JRE (*Java Runtime Environment*) solo contiene las herramientas necesarias para ejecutar aplicaciones Java, como son la Maquina Virtual de Java, las bibliotecas de Java entre otros. Así, si se requiere realizar programas en Java, es necesario el JDK, pero si solo se requiere ejecutar un programa, por ejemplo un *applet* de Java, basta con instalar el JRE.

Para todos las aplicaciones Java desarrolladas en la presente tesis se usó Java Development Kit 6.24, como entorno de desarrollo para Java, en un inicio se usó NetBeans IDE[5] y después se optó por BlueJ[6]. En C++ como entorno de desarrollo se usó Code::Blocks IDE[7] y Visual C++ 2010 Express[8]. Todas estas tecnologías son gratuitas y están disponibles tanto para Windows como Linux a excepción del software de Microsoft.

III.2. C/C++ y JNI

Como se a explicado, Java es por sí solo un lenguaje muy bueno para crear aplicaciones completas. Pero en ocasiones hay tareas que requieren llamar a código ajeno a Java, denominado código nativo. Las razones son varias, por ejemplo si se desea acceder a aspectos específicos del sistema operativo, como el registro de Windows, interactuar con dispositivos

[4]http://www.oracle.com/technetwork/indexes/downloads/index.html
[5]Sitio web oficial http://www.netbeans.org/
[6]Sitio web oficial http://www.bluej.org/
[7]Sitio web oficial http://www.codeblocks.org/
[8]Sitio web oficial http://www.microsoft.com/visualstudio/

hardware especiales, tal como un puerto serie, reutilizar código ya existente en otro lenguaje, o la necesidad de maximizar la velocidad del código(Horstmann and Cornell (2006b)).

Para lograr esto, Java cuenta con una interfaz dedicada a facilitar la interacción con el código nativo. Se conoce como JNI (*Java Native Interface*), el cual enlaza código Java con código nativo en C o C++ únicamente. Con JNI se pueden invocar funciones en C/C++ desde el lenguaje de programación Java, crear, inspeccionar y actualizar objetos Java, capturar y enlazar excepciones, cargar y obtener información de clases de Java en C/C++, entre otras tareas. Para información básica sobre el tema de esta sección se recomienda el capitulo 11 de Horstmann and Cornell (2006b) y para consultas más a fondo el libro de Liang (1999).

Invocar a un método nativo.

El ejemplo mas simple es invocar un método nativo que imprima desde Java el típico mensaje "Hola Mundo" por medio de printf, la cual es una función de la biblioteca estándar de C. Para iniciar, se crea un proyecto Java con dos clases. La clase principal se llama MostrarMensaje y la clase que contiene al método nativo se llama Mensaje. El código se muestra en los ejemplos 3.1 y 3.2, respectivamente.

Ejemplo 3.1: MostrarMensaje.java

```
1 public class MostrarMensaje
2 {
3     public static void main(String[] args)
4     {
5         Mensaje.saludo("Generado con JNI") ;
6     }
7
8 }
```

Ejemplo 3.2: Mensaje.java

```
1 public class Mensaje
2 {
3     public static native void saludo(String msj);
4 }
```

Observemos que el método de la clase Mensaje se declara como `native`, esta palabra es reservada en Java y se usa para denotar los métodos nativos, anunciando al compilador que ese método se va a definir externamente. Este código, por el momento, solo puede ser compilado pero no ejecutado. Se compila Mensaje.java y esta operación nos da como resultado el archivo Mensaje.class(código en *bytecode* que interpreta la maquina virtual). A continuación, es necesario ejecutar *javah* sobre el archivo *class*, especificando la opción `jni`:

```
javah -jni Mensaje
```

El resultado de esta operación es un archivo de encabezado nativo que se muestra en el ejemplo 3.3, el cual contiene la función correspondiente en C del método declarado anteriormente. Si existiera mas de un método declarado como nativo en la clase, entonces, el archivo de encabezado tendría una declaración de una función por cada uno.

Ejemplo 3.3: Mensaje.h

```
1  /* DO NOT EDIT THIS FILE - it is machine generated */
2  #include <jni.h>
3  /* Header for class Mensaje */
4
5  #ifndef _Included_Mensaje
6  #define _Included_Mensaje
7  #ifdef __cplusplus
8  extern "C" {
9  #endif
10 /*
11  * Class:     Mensaje
12  * Method:    saludo
13  * Signature: (Ljava/lang/String;)V
14  */
15 JNIEXPORT void JNICALL Java_Mensaje_saludo
16   (JNIEnv *, jclass, jstring);
17
18 #ifdef __cplusplus
19 }
20 #endif
21 #endif
```

La función ha sido nombrada `Java_Mensaje_saludo`. Además, se incluye el archivo de encabezado *jni.h* que provee información necesaria al código nativo para llamar funciones de la interfaz nativa de Java. También se puede observar que la función tiene dos argumentos que no fueron especificados en la clase. Estos argumentos siempre son incluidos en todas las llamadas a métodos nativos. El primer argumento es un apuntador a una tabla de apuntadores a funciones de JNI. El segundo argumento puede variar dependiendo de si se trata de un método estático o un método no estático. Al ser este un método estático, el segundo argumento es de tipo `jclass` el cual es una referencia de la clase. Si se tratara de un método no estático recibiría una referencia del argumento implícito `this`, el cual sería de tipo `jobject`. El uso de estos argumentos se explicara con mayor detalle mas adelante.

El siguiente paso es implementar el método nativo, como se muestra en el ejemplo 3.4.

Ejemplo 3.4: Mensaje.c

```
1  #include <jni.h>
2  #include <stdio.h>
3  #include "Mensaje.h"
4
5  JNIEXPORT void JNICALL Java_Mensaje_saludo
6          (JNIEnv *env, jclass cl, jstring jMsg)
7  {
8      const char* msg ;
```

```
9       msg=(*env)-> GetStringUTFChars (env,jMsg,NULL ) ;
10      printf ("Hola Mundo: %s\n", msg) ;
11      (*env)-> ReleaseStringUTFChars (env,jMsg, msg) ;
12  }
```

Si se desea trabajar en C++ es necesario anteponer `extern "C"` a la declaración de las funciones que implementen los métodos nativos. Esto evita que el compilador genere código especifico de C++.

Se crea un proyecto de biblioteca de enlace dinámico que incluya este archivo y `Mensaje.h`. Si se trabaja en Linux, al proyecto se le deben agregar los siguientes directorios:

```
jdk/include/
jdk/include/linux
```

Si es en Windows, son los siguientes:

```
jdk\include\
jdk\include\win32
```

El segmento jdk debe ser sustituido por el directorio completo de la ubicación de la versión del JDK instalado. Al compilar se obtiene un archivo de extensión .so en Lnux y .dll en Windows. A continuación, en Windows es necesario modificar la variable PATH agregando el directorio donde se encuentra esta biblioteca. Hecho esto, lo siguiente es modificar Mensaje.java como en el ejemplo 3.5.

Ejemplo 3.5: Mensaje.java

```
1  public class Mensaje
2  {
3      public static native void saludo(String msj);
4      static
5      {
6          System.loadLibrary("libMensaje");
7      }
8  }
```

Lo agregado es un inicializador `static`, el cual se encarga de cargar automáticamente la biblioteca nativa al usar la clase por primera vez. En Linux es mejor agregar el siguiente fragmento de código y no es necesario agregar ninguna ruta al PATH.

```
static
{
   System.load("ruta_de_acceso/libMensaje.so");
}
```

Por ultimo, ejecutamos el programa y debe aparecer la frase **Hola Mundo: Generado con JNI**.

Correspondencia entre tipos de datos.

En una sección anterior, se menciono que en C existe un problema con el tamaño de los tipos de datos por la dependencia del sistema operativo. Para lograr una buena interacción entre Java y el código nativo se definen en el archivo *jni.h* la correspondencia de tipos de Java con los de C. La tabla 3.1 muestra esta correspondencia.

Tabla 3.1: Correspondencia de tipos de Java con tipos de C

Lenguaje de programación Java	Lenguaje de programación C	Bytes
boolean	jboolean	1
byte	jbyte	1
char	jchar	2
short	jshort	2
int	jint	4
long	jlong	8
float	jfloat	4
double	jdouble	8

En Java también existen arreglos de tipos los cuales cuentan con sus respectivos tipos en C como se muestra en la tabla 3.2.

Tabla 3.2: Correspondencia de arreglos de Java con arreglos de C

Tipo en Java	Tipo en C
boolean[]	jbooleanArray
byte[]	jbyteArray
char[]	jcharArray
int[]	jintArray
short[]	jshortArray
long[]	jlongArray
float[]	jfloatArray
double[]	jdoubleArray
Object[]	jobjectArray

Accediendo a funciones JNI.

Como se menciono entes, todo método nativo recibe un apuntador de tipo `JNIEnv`, el cual apunta a un arreglo de apuntadores a funciones de JNI. Así un método nativo puede tener acceso a este gran conjunto de funciones por medio de este apuntador. En C es necesario anteponer `(*env)->` a cualquier llamada de una función JNI. Por otro lado, en C++ es más fácil llamar a una función JNI, pues solo es necesario anteponer `env->` y se omite el apuntador `JNIEnv` en la lista de argumentos de la llamada. Por ejemplo, en C++ se puede llamar a la función `GetStringUTFChars` en la forma

```
msg=env->GetStringUTFChars ( jMsg, NULL) ;
```

Las funciones de JNI pueden clasificarse en categorías. A continuación se explican algunas tareas básicas de estas funciones por su categoría.

Llevar a cabo operaciones de cadenas y arreglos. En el ejemplo Mensaje.c se muestra como acceder a una cadena de Java. Estas están en formato Unicode y para ser usadas en funciones de C es necesario convertirlas a caracteres ASCII con la función

```
const jbyte* GetStringUTFChars(JNIEnv* env,
jstring string, jboolean* isCopy);
```

donde `jstring` es la cadena con la que se desea trabajar y el argumento `isCopy` determina si el puntero devuelto es el arreglo original, o si se ha hecho una copia de éste. Se proporciona un puntero de la codificación en "UTF-8 modificado" de una cadena, o bien `NULL` si no es posible construir la matriz de caracteres. El puntero esta disponible hasta que se haga una llamada a

```
void ReleaseStringUTFChars(JNIEnv* env, jstring string,
const char* utf_buffer);
```

para notificar a la máquina virtual que el método nativo ya no necesita acceder a la cadena de Java. Para convertir una cadena de caracteres de C a una cadena Java se llama a la función

```
jstring NewStringUTF(JNIEnv* env,const char* bytes);
```

la cual proporciona un nuevo objeto de cadena de Java o `NULL` si no es posible construir la cadena.

También se cuenta con funciones JNI para manipular arreglos. Por ejemplo, la función

```
jsize GetArrayLength(JNIEnv* env, jarray array);
```

proporciona la longitud de una matriz. Para arreglos de tipos primitivos, las funciones

```
jxxx* GetXxxArrayElements(JNIEnv* env,
jxxxArray array, jboolean* isCopy);
```

(donde `Xxx` es un tipo primitivo, por ejemplo, si se tiene un arreglo de tipo `double` se tiene la función `jdouble* GetDoubleArrayElement`), proporciona un putero en C que señala al elemento inicial de la matriz, entonces se pueden leer y escribir directamente los elementos de esta. Para garantizar la realización de los cambios y para indicar a la maquina virtual que ya no se necesita el apuntador, se llama a la función correspondiente

```
void ReleaseXxxArrayElements(JNIEnv* env,
jxxxArray array, jboolean* buffer, jint mode);
```

Para acceder a los elementos de una matriz de referencias de objetos se emplean los métodos

```
jobject GetObjectArrayElement(JNIEnv* env,
jobjectArray array, jsize index);
void SetObjectArrayElement(JNIEnv* env,
jobjectArray array, jsize index, jobject value);
```

La primera proporciona el valor de un elemento de la matriz y la segunda da un nuevo valor a un elemento de la matriz. Para crear un arreglo de referencias se cuenta con la función

```
jarray NewObjectArray(JNIEnv* env, jsize length,
jclass elementType, jobject initialElement);
```

El parámetro `initialElement` es el valor inicial al que se fijan todos los elementos del arreglo, y puede ser `NULL`. El parámetro `elementType` indica el tipo de los elementos del arreglo, y no puede ser `NULL`.

Acceder a campos de ejemplar o estáticos. En los ejemplos 3.6, 3.7 y 3.8 se muestra como acceder a campos de ejemplar, en particular al campo `sueldo` de la clase `Empleado` desde el método nativo `subirSueldo`.

Ejemplo 3.6: PruebaEmpleado.java

```
1  public class PruebaEmpleado
2  {
3     public static void main(String[] args)
4     {
5        Empleado[] personal = new Empleado[3];
6        personal[0] = new Empleado(
7             "Jennifer Nieto",
8             45000 );
9        personal[1] = new Empleado(
10            "Victor Rosas",
11            23000 );
12       personal[2] = new Empleado(
13            "Josefina Barcenas",
14            37000 );
15       int i;
16       for (Empleado e : personal)
17            e.subirSueldo(5);
18       for (Empleado e : personal)
19            e.print();
20    }
21 }
```

Ejemplo 3.7: Empleado.java

```
1  public  class Empleado
2  {
3     public Empleado(String n, double s)
4     {
5        nombre = n;
6        sueldo = s;
7     }
8     public native void subirSueldo(double porcentaje);
9     public void print()
10    {
11       System.out.println(nombre + " " + sueldo);
12    }
13    private String nombre;
14    private double sueldo;
15    static
16    {
17       System.loadLibrary("Empleado");
18    }
19 }
```

Ejemplo 3.8: Empleado.c

```
1  #include "Empleado.h"
2  #include <stdio.h>
3  JNIEXPORT void JNICALL Java_Empleado_subirSueldo
4  (JNIEnv* env, jobject obj_this, jdouble porcentaje)
5  {
6     /*se obtiene la clase*/
7     jclass class_Empleado = (*env)->GetObjectClass(env,
8     obj_this);
9     /*se obtiene la ID del campo*/
10    jfieldID id_sueldo = (*env)->GetFieldID(env,
11    class_Empleado, "sueldo", "D");
12    /*se obtiene el valor del campo*/
13    jdouble sueldo = (*env)->GetDoubleField(env, obj_this,
14    id_sueldo);
15    sueldo *= 1 + porcentaje / 100;
16    /*se especifica el valor del campo*/
17    (*env)->SetDoubleField(env, obj_this, id_sueldo,
18    sueldo);
19 }
```

Como se puede observar el método `subirSueldo` no se declaro `static`, por lo tanto cuando se define en C el segundo argumento es de tipo `jobject`. Este es una referencia del argumento implícito `this`, el cual permitirá el acceso al campo `sueldo`. Para acceder a campos de ejemplar primero se debe obtener la clase a la cual pertenece el objeto que contiene al campo, lo cual se logra llamando a la función

```
jclass GetObjectClass(JNIEnv *env, jobject obj);
```

con la referencia de un objeto que pertenece a la clase como argumento. Lo siguiente es obtener el identificador del campo con la función

```
jfieldID GetFieldID(JNIEnv *env, jclass class,
const char *name, const char *sig);
```

a la cual se debe aportar, además de el apuntador `env` y la clase, el nombre del campo y su signatura. La signatura de un campo es una codificación del tipo de dato correspondiente. Los métodos también tienen su signatura, la cual describe sus parámetros y el tipo proporcionado por este. Para generar la signatura de los métodos y campos de una clase se puede utilizar la orden `javap` con la opción `-s` sobre el archivo compilado que contiene la clase. Por ejemplo,

```
javap -s -private Empleado
```

Proporciona el resultado siguiente.

```
Compiled from "Empleado.java"
public class Empleado extends java.lang.Object{
private java.lang.String nombre;
  Signature: Ljava/lang/String;
private double sueldo;
  Signature: D
public Empleado(java.lang.String, double);
  Signature: (Ljava/lang/String;D)V
public native void subirSueldo(double);
  Signature: (D)V
public void print();
  Signature: ()V
static {};
  Signature: ()V
}
```

Para obtener el valor del campo se llama la función

```
NativeType GetXxxField(JNIEnv *env, jobject obj,
jfieldID fieldID);
```

donde `Xxx` puede ser `Object` o cualquier tipo primitivo. Se deben especificar tres argumentos: el apuntador `env`, la referencia de tipo `jobject` y el identificador del campo. Para dar un nuevo valor a un campo se llama a la función

```
void SetXxxField(JNIEnv *env, jobject obj,
jfieldID fieldID, NativeType value);
```

con los tres argumentos anteriores y el nuevo valor del campo.

En el caso en el cual se desee tener acceso a campos estáticos existen ciertas diferencias. Primero al no tener un objeto, hay que utilizar la función

```
jclass FindClass(JNIEnv *env, const char *name);
```

en lugar de `GetObjectClass` para obtener la referencia de la clase, en el segundo argumento se debe especificar el nombre de la clase. Para obtener el identificador del campo se cuenta con la función respectiva

```
jfieldID GetStaticFieldID(JNIEnv *env, jclass class_name,
const char *name, const char *sig);
```

cuyos argumentos son la clase, en nombre del campo y la signatura. Además, el programador es quien debe proporcionar la clase cuando se accede a los campos mediante las funciones

```
NativeType GetStaticXxxField(JNIEnv *env, jclass clazz,
jfieldID fieldID);
void SetStaticXxxField(JNIEnv *env, jclass clazz,
jfieldID fieldID, NativeType value);
```

para obtener y establecer valores respectivamente.

Llamar a métodos de instancia y estáticos. Para llamar a métodos de instancia son necesarios tres pasos. Primero se obtiene la clase con la función `GetObjectClass` ya antes descrita. El segundo paso es obtener el identificador del método con la función

```
jmethodID GetMethodID(JNIEnv* env, jclass class,
const char* name, const char* signature);
```

aportando la clase, el nombre del método y la signatura de método. El tercer paso es llamar al método con la función

```
xxx CallXxxMethod(JNIEnv* env, jobject object,
jmethodID methodID, ...);
```

donde `Xxx` se sustituye por `Void`, `Int`, `Object`, etc., dependiendo del tipo proporcionado por el método. Se deben aportar como argumentos el parámetro implícito, el identificador del método y los parámetros explícitos que pueden variar dependiendo del método.

El caso de métodos estáticos es un proceso similar. Primero, como no se cuenta con un objeto de la clase, se utiliza `FindClass`. A continuación, es necesario obtener el identificador del método estático por medio de la función

```
jmethodID GetStaticMethodID(JNIEnv* env, jclass class,
const char* name, const char* signature);
```

al cual se le proporcionan como argumentos el apuntador de la interfaz JNI, la clase, el nombre del método y la signatura codificada del método. Por ultimo, se puede hacer la llamada del método con la función

```
xxx CallStaticXxxMethod(JNIEnv* env, jclass class,
jmethodID methodID,...);
```

donde `Xxx` depende del tipo que proporciona el método. Los argumentos que se deben proporcionar son el apuntador de la interfaz JNI, la clase, el identificador del método y los argumentos del método.

Llevar a cabo operaciones de clase y objetos. Los métodos nativos también pueden crear objetos de Java. Primero se obtiene la clase como se vio anteriormente. Después se necesita obtener el identificador del método constructor mediante `GetMethodID`, con un nombre de método igual a "<init>" y un tipo proporcionado `void`. Lo siguiente es llamar a la función

```
jobject NewObject(JNIEnv, jclass class,
jmethodID constructorID,...);
```

y proporcionarle como argumentos el apuntador de la interfaz JNI, la clase de la que se crea un ejemplar, el identificador y los argumentos del constructor que pueden variar.

Obtener información de la versión. La función

```
jint GetVersion(JNIEnv *env);
```

proporciona información sobre la versión de JNI que se utilize, la versión es devuelta en formato de 32 bits, los primeros 16 bits son para el primer número de la versión y los otros 16 bits son el segundo número de versión.

Generar y gestionar excepciones Java. Con JNI es posible manejar excepciones igual que si se trabajara en Java. Hay que hacer notar que en C no existen las excepciones. Sin embargo, C++ si cuenta con excepciones propias, por lo que si se trabaja en este y con JNI, hay que asegurarse de que el método nativo no lance excepciones de C++ si ya

existen de JNI. Cuando se produce una excepción no se interrumpe el flujo normal de ejecución del programa (como pasa en Java) sino que en el hilo se indica que se ha lanzado la excepción. Existe un único indicador por cada hilo y éste está almacenado en la estructura `env`, con lo que el hecho de que se produzca una excepción en un hilo no afecta a los demás hilos. Las funciones para gestión de excepciones son:

- `jint Throw(JNIEnv* jthrowable)`. Lanza un objeto de excepción ya existente `jthrowable`. Se usa en los métodos nativos para relanzar una excepción.
- `jint ThrowNew(JNIEnv* env, jclass class, const char* message)`. Genera un nuevo objeto de excepción lo cual activa el indicador y deja la excepción pendiente hasta que sea tratada. La función retorna 0 si la excepción se lanza o un valor distinto de 0 si no se pudo lanzar la excepción (por ejemplo porque ya había otra lanzada o porque la clase no deriva de `Throwable`).
- `jthrowable ExceptionOccurred(JNIEnv* env)`. Determina si hay alguna excepción pendiente en el hilo y devuelve una referencia a la excepción ocurrida o `NULL` si no ha habido excepción.
- `jboolean ExceptionCheck(JNIEnv* env)`. Determina si hay alguna excepción pendiente en el hilo.
- `void ExceptionClear(JNIEnv* env)`. Esta función hace lo mismo que el bloque `catch` de Java, que también desactiva la excepción una vez tratada.
- `void ExceptionDescribe(JNIEnv* env)`. imprime una traza de la pila de llamadas a funciones en `System.err`. Un efecto lateral es que desactiva el indicador de la excepción pendiente.

Si no se trata de forma adecuada las excepciones los resultados pueden ser impredecibles.

El API de invocación.

El API de invocación permite colocar la maquina virtual de Java en cualquier programa escrito en C o C++. El código del ejemplo 3.9 activa a la máquina virtual y después llama al método `main` de la clase `HolaMundo` del ejemplo 3.10.

Ejemplo 3.9: PruebaInvocacion.c

```
1  #include <jni.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  #define USER_CLASSPATH "ruta/de/la/clase/Java"
7
8  int main(int argc, char** argv) {
9
10 ////Código para condiciones
```

```
11 ////iniciales de la MV->
12     JavaVMInitArgs vm_args;
13     JavaVMOption options[1];
14     JavaVM *jvm;
15     JNIEnv *env;
16
17     options[0].optionString =
18         "-Djava.class.path="USER_CLASSPATH;
19
20     memset(&vm_args, 0, sizeof(vm_args));
21     vm_args.version = JNI_VERSION_1_2;
22     vm_args.nOptions = 1;
23     vm_args.options = options;
24
25     if ( JNI_CreateJavaVM(&jvm, (void**) &env,
26         &vm_args)!= JNI_OK)
27     {
28        printf(
29         "Error al crear la maquina virtual\n");
30        return (*jvm)->DestroyJavaVM(jvm);
31     }
32 ////<-Código para condiciones
33 ////iniciales de la MV
34
35 ////Acceso normal de código
36 ////Java desde C ->
37     jobjectArray args;
38     jclass class_Proof =
39       (*env)->FindClass(
40           env,
41           "HolaMundo");
42     if(class_Proof == 0){
43       fprintf(stderr,
44           "Clase no encontrada.\n");
45       return -1;
46     }
47
48     jmethodID id_main =
49       (*env)->GetStaticMethodID(
50         env,
51         class_Proof,
52         "main",
53         "([Ljava/lang/String;)V");
54     jclass class_String =
55     (*env)->FindClass(
```

```
56          env,
57          "java/lang/String");
58      args = (*env)->NewObjectArray(
59          env,
60          0,
61          class_String,
62          NULL);
63      (*env)->CallStaticVoidMethod(
64          env,
65          class_Proof,
66          id_main,
67          args);
68  ////<-Acceso normal de código
69  ////Java desde C
70
71  ////Termina la ejecución de
72  ////la máquina virtual
73      return (*jvm)->DestroyJavaVM(jvm);
74  }
```

Ejemplo 3.10: HolaMundo.java

```
1  public class HolaMundo
2  {
3      public static void main(String[] args)
4      {
5          String[] saludo = new String[3];
6          saludo[0] =
7          "****************************************";
8          saludo[1] =
9          "*** Hola Mundo Java invocado desde C ***";
10         saludo[2] =
11         "****************************************";
12         for (String g : saludo)
13             System.out.println(g);
14     }
15 }
```

El ejemplo 3.10 es una clase cuyo método principal imprime en pantalla cadenas de caracteres sin mayor dificultad. En cambio, el ejemplo 3.9 es mucho más interesante. Primero se incluyen algunas bibliotecas de C y el archivo de encabezado jni. A continuación se define una cadena que contiene la ruta de acceso al archivo `HolaMundo.class`. En el método principal se señala el código mínimo que se necesita para poner en condiciones iniciales una máquina virtual. Se declara una estructura `JavaVMInitArgs` nombrada como `vm_args` la cual contendrá información para inicializar la maquina virtual y tiene la siguiente forma:

```
typedef struct JavaVMInitArgs {
```

```
jint version;
jint nOptions;
JavaVMOption *options;
jboolean ignoreUnrecognized;
} JavaVMInitArgs;
```

El campo `version` debe tener el valor `JNI_VERSION_1_2`, el campo `nOptions` indica el número de opciones que se pasaran en `options` y el campo `ignoreUnrecognized` si vale `JNI_TRUE` indica que se pueden ignorar las opciones no estándar (en unas máquinas virtuales se puedan pasar opciones adicionales propias de esa implementación de máquina virtual) si la máquina virtual que estamos usando no entiende alguna de ellas. El campo `options` es un puntero a un arreglo de elementos del tipo `JavaVMOption` que tienen la siguiente forma:

```
typedef struct JavaVMOption {
char *optionString;
void *extraInfo;
} JavaVMOption;
```

Las opciones más usadas que se pueden colocar en `optionString` son

- `-D<name>=<value>` Fija una propiedad del sistema.
- `-verbose` Habilita la salida de información de lo que está haciendo.

Continuando con la descripción del ejemplo, se declara el apuntador `jvm` de tipo `JavaVM` donde se depositara un apuntador a la máquina virtual. Este se usara para referirse a la instancia de la máquina virtual.

En las siguientes lineas se establecen las opciones iniciales ya descritas para la máquina virtual hasta llegar a la creación de la máquina virtual por medio de la función

```
jint JNI_CreateJavaVM(JavaVM** jvm, void** env,
void* vm_args)
```

La cual tiene como argumentos de entrada el apuntador `jvm`, el apuntador `env` a funciones JNI y la estructura `vm_args` con las opciones iniciales. La función retorna cero si tiene éxito o un número negativo en caso contrario. Esta función no es parte de la tabla de funciones de JNI, por esta razón se puede acceder sin el apuntador `env`. Una función importante es

```
jint DestroyJavaVM(JavaVM *vm);
```

la cual descarga la máquina virtual y reclama todos los recursos. El hilo que llama a esta función queda bloqueado hasta que es el único que queda vivo. Esta restricción existe porque los demás hilos pueden tener asignados recursos (por ejemplo una ventana). La función retorna cero si tiene éxito o un número negativo en caso contrario.

En el resto del ejemplo se realiza el proceso para llamar código Java desde C como se explico anteriormente.

Para compilar el proyecto es necesario incluir los directorios correspondientes ya antes mencionados. Además, en el caso de Linux, se agrega la biblioteca de enlace dinámico `libjvm.so` que se encuentra en `jdk/jre/lib/i386/client`.

En el caso de Windows, se agrega la biblioteca de enlace dinamico jvm.lib que se encuentra en `jdk\lib` y el directorio `jdk\jre\bin\client` se agrega al PATH.

III.3. Swing

Desde Java 1.0 se contaba con una biblioteca para programar interfaces gráficas de usuario, denominada *Abstract Window Toolkit* (AWT). Se basaba en crear herramientas equivalentes a las de la interfaz de usuario nativo. Esta biblioteca resultó ser muy complicada de crear, pues si se deseaba una GUI compleja y transportable, el resultado era poco coherente e impredecible por las diferencias que existen entre las interfaces de usuario nativas.

Para Java 1.2 se agregó Swing como biblioteca estándar para GUI. Swing hace uso de una metodología distinta al de AWT. La única funcionalidad equivalente que se necesita es la forma de sacar ventanas y de pintar sobre ellas, pues los elementos de la interfaz de usuario, como botones, menús y demás, se pintan sobre ventanas en blanco. De esta forma, las interfaces de usuario tienen el mismo aspecto y comportamiento independientemente de la plataforma en que se este ejecutando el programa. Sin embargo, Swing no es un sustituto completo de AWT, solo ofrece componentes de interfaz de usuario mas potentes y AWT cuenta con herramientas indispensables como el manejo de eventos (Horstmann and Cornell (2006a)).

Una aplicación con GUI necesita de una ventana de nivel superior donde se colocan los demás componentes, como lo son botones, menús, áreas de texto, etc. Estos componentes en Java se conocen como marcos y son de los pocos que dependen del sistema de ventanas del sistema nativo. La clase de los marcos se llama `JFrame` y la manera de crearlos y visualizarlos es sencilla como se puede observar en el ejemplo 3.11.

Ejemplo 3.11: PruebaMarcoSencillo.java

```
1  import javax.swing.*;
2  public class PruebaMarcoSencillo
3  {
4      public static void main(String[] args)
5      {
6          MarcoSencillo marco = new MarcoSencillo();
7          marco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8          marco.setVisible(true);
9      }
10 }
11
12 class MarcoSencillo extends JFrame
13 {
14     public MarcoSencillo()
15     {
16         setSize(ANCHURA_PREFIJADA, ALTURA_PREFIJADA);
```

```
17      }
18      public static final int ANCHURA_PREFIJADA=300;
19      public static final int ALTURA_PREFIJADA=200;
20  }
```

En la primera clase , `PruebaMarcoSencillo`, se crea un nuevo objeto de tipo `MarcoSencillo`, el cual extiende a la clase `JFrame`. Los marcos tienen un tamaño inicial de 0 x 0 lo cual no se desea, es por eso que se definen una altura y anchura de 300 x 200 y se establecen con la función `setSize`. Además, en el método principal, se establece la operación que debe realizar cuando se cierra la ventana, la cual en este caso es concluir la aplicación. Por ultimo, se ordena que el marco sea visible, como se muestra en la figura 3.1.

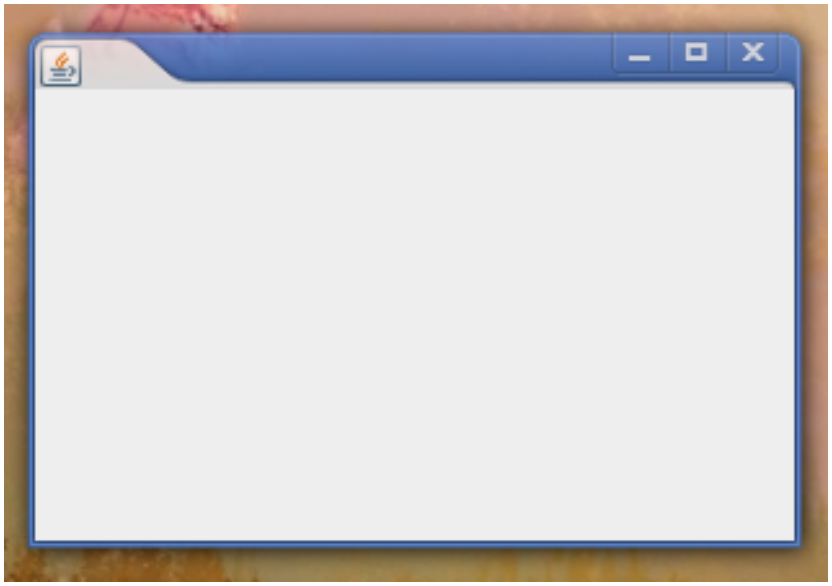


Figura 3.1: Ejemplo de un marco.

III.3.1 Componentes

Un marco por si solo carece de funcionalidad, lo interesante es agregar componentes que realicen alguna tarea. La clase *JFrame* cuenta con el método `add` el cual recibe un ejemplar de un objeto que extienda a la clase *Component* y lo agrega. Múltiples clases cuentan con este método y sus variantes, además existen otros métodos para agregar diferentes características. Aunque se pueden agregar toda clase de componentes de forma directa sobre el marco, es considerado como una mejor practica el agregar primero componentes contenedores llamados láminas, en los cuales se colocan otros componentes, inclusive más láminas. La clase de las láminas es *JPanel* y la forma de trabajar con ellas es como se muestra a continuación:

```
JFrame marco = new JFrame("Ejemplo");
JPanel lamina = new JPanel();
lamina.add(new JButton());
marco.add(lamina);
```

Se puede observar como primero se crea un ejemplar de la lámina, se agregan los componentes y por ultimo se coloca en el marco correspondiente. A continuación se enlistan algunos componentes de la interfaz gráfica de usuario basados en Swing:

JPanel Lámina contenedora de otros componentes.

Constructores

JPanel Las variantes son:

- public JPanel()
- public JPanel(LayoutManager layout)

Parámetros:

`layout` - encargado de disposición.[1]

JFrame Marco principal para una aplicación de escritorio.

Constructores

JFrame Las variantes son:

- public JFrame()
- public JFrame(String title)

Crea un marco inicialmente invisible con el titulo especificado.

Parámetros:

`title` - titulo del marco.

Métodos

setSize public void setSize(int width, int height)

Establece las dimensiones del marco. Un marco tiene por omisión las dimensiones de 0 x 0.

Parámetros:

`width` - anchura del marco en píxeles.

`height` - altura del marco en píxeles.

setUndecorated public void setUndecorated(boolean undecorated)

Activa o desactiva la decoración del marco.

Parámetros:

`undecorated` - si es `true` activa la decoración. Si es `false` la desactiva.

setDefaultCloseOperation public void setDefaultCloseOperation(int operation)

Establece la operación que debe realizar el marco al cerrarse.

Parámetros:

`operation` - valor de la operación que se debe realizar al cerrar el marco. Lo más común es establecer el salir de la aplicación, cuyo valor es la constante `JFrame.EXIT_ON_CLOSE`.

setLayout public void setLayout(LayoutManager manager)

Establece el encargado de disposición del marco.[9]

[9] Véase la subsección *Encargados de disposición*

Parámetros:

`manager` - el encargado de disposición.

setVisible public void setVisible(boolean b)

Muestra u oculta el marco, dependiendo del parámetro.

Parámetros:

`b` - si es `true` el marco es visible, si es `false` se oculta.

JButton. Botón común al cual se le puede rotular y establecer un icono.

```
JButton boton = new JButton("<html><B>Nombre</B></html>",
                            new ImageIcon("boton.jpg"));
```

Constructores

JButton Las variantes son:

- public JButton()
- public JButton(Icon icon)
- public JButton(String text)
- public JButton(String text, Icon icon)

Parámetros:

`icon` - icono de botón.

`text` - rótulo del botón. Se admiten cadenas HTML para darle formato al texto.

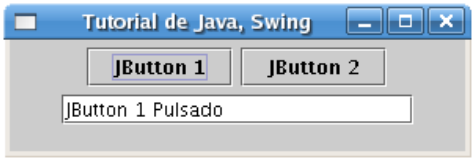


Figura 3.2: Ejemplo de un botón en java.

JCheckBox. Son casillas de verificación para los casos en que es necesario tratar con entradas del tipo "si" o "no". Al igual que los botones, se pueden establecer tanto el rótulo como un icono y, además, si inicia seleccionado o no. Se puede determinar si una casilla se encuentra seleccionada con el método `isSelected`, el cual retorna `true` si lo esta o `false` si no.

```
JCheckBox boton = new JCheckBox("Nombre",
                        new ImageIcon("check.jpg"),
                        true);
```

Constructores

JCheckBox Las variantes son:

- public JCheckBox()
- public JCheckBox(Icon icon, boolean selected)
- public JCheckBox(String text, boolean selected)
- public JCheckBox(String text, Icon icon, boolean selected)

`icon` - icono de botón.
`text` - rótulo del botón. Se admiten cadenas HTML para darle formato al texto.
`selected` - si es `true` la casilla aparece seleccionada y si es `false` no.

Figura 3.3: Ejemplo de una casilla de verificación en java.

JRadioButton. Los botones de radio se organizan en grupos con la posibilidad de seleccionar solo uno de ellos. Para crear un grupo de botones de radio se debe crear un objeto de la clase `ButtonGroup` al cual se irán agregando cada uno de los elementos que pertenecerán al grupo. Se puede especificar a cada botón el rótulo, icono y si está seleccionado o no, pero con la condición de que solo uno sera verdadero y todos los demás falso.

```
ButtonGroup grupo = new ButtonGroup();
JRadioButton boton1 = new JRadioButton("Nombre1",
  new ImageIcon("radio1.jpg"), true);
grupo.add(boton1);
JRadioButton boton2 = new JRadioButton("Nombre2",
  new ImageIcon("radio2.jpg"), false);
grupo.add(boton2);
...
```

Constructores

JRadioButton Las variantes son:

- public JRadioButton()
- public JRadioButton(Icon icon, boolean selected)
- public JRadioButton(String text, boolean selected)

- public JRadioButton(String text, Icon icon, boolean selected)

Parámetros:

`icon` - icono de botón.

`text` - rótulo del botón. Se admiten cadenas HTML para darle formato al texto.

`selected` - si es `true` el botón de radio aparece seleccionada y si es `false` no.

Figura 3.4: Ejemplo de un botón de radio en java.

Border. Cuando se tienen diferentes grupos de botones de radio y otros componentes, resulta apropiado señalar esta distribución. Los bordes cumplen con este objetivo, delimitan cada grupo y además se puede agregar un título. Los bordes se pueden aplicar a cualquier componente que extienda a la clase `JComponent` mediante una llamada al método `setBorder`.

```
Border borde = BorderFactory.createEtchedBorder();
Border conTitulo = BorderFactory.createTitleBorder(
  grabado,"Título");
lamina.setBorder(conTitulo);
```

JLabel. Etiquetas para colocar, por lo general, al lado de componentes sin rótulo, por ejemplo en los campos de texto.

```
Jlabel etiqueta = new JLabel("Nombre");
```

Constructores

JLabel public JLabel(String text)

Parámetros:

`text` - rótulo del botón. Se admiten cadenas HTML para darle formato al texto.

JTextField. Los campos de texto tienen la función de capturar entradas del usuario que solo necesiten de una línea de texto. Carecen de rótulo propio por lo que se auxilia de una etiqueta en los casos que así lo requieran. Se puede establecer el número de columnas que tendrá de ancho el campo, así como el texto que aparecerá por defecto. Es importante ver que el número de columnas no es un límite superior del número de caracteres, si el texto excede la zona visible la entrada se desplaza. La clase cuenta con la función `getText`, que proporciona el texto que se encuentre en el campo en forma de una cadena.

```
JTextField campoDeTexto = new JTextField("Texto",20);
```

Constructores

JTextField Las variantes son:

- public JTextField()
- public JTextField(String text)
- public JTextField(int columns)
- public JTextField(String text,int columns)

Parámetros:

`text` - la cadena que se despliega inicialmente.
`columns` - número de columnas de ancho.

Métodos

setEditable public void setEditable(boolean b)
Establece si el campo es editable o no.

Parámetros:

`b` - si es `true` el campo es editable, si es `false` no.

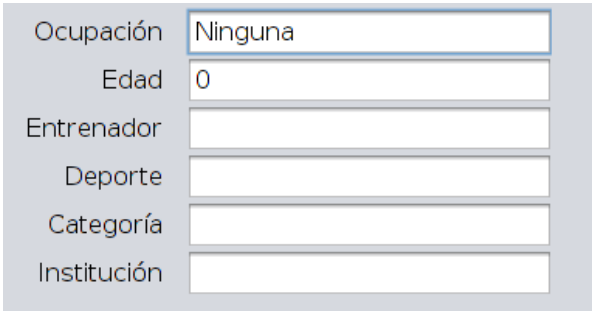


Figura 3.5: Ejemplo de un campo de texto junto con su etiqueta en java.

JTextArea. Las áreas de texto pueden capturar múltiples lineas a diferencia de los campos. Se puede especificar el numero de filas y columnas de la zona de texto. Por defecto, el texto se recorta si excede la zona visible. Existen dos alternativas para evitar este comportamiento, la primera es activar los saltos de línea automáticos llamando a la función `setLineWrap` con el argumento `true`. Estos saltos de línea son solo un efecto visual, no se agregan caracteres '\n' en el texto. La segunda consiste en agregar el área de texto dentro de una lámina con desplazamiento para contar con barras de desplazamiento.

```
JTextArea zonaTexto = new JTextArea(
    new PlainDocument(),"texto",8,40);
JScrollPane laminaDesplazamiento =
    new JScrollPane(zonaTexto);
```

Constructores

JTextArea Las variantes son:

- public JTextArea()
- public JTextArea(String text)
- public JTextArea(int rows, int columns)
- public JTextArea(Document doc)
- public JTextArea(String text, int rows, int columns)
- public JTextArea(Document doc, String text, int rows, int columns)

Parámetros:

`text` - la cadena que se despliega inicialmente.

`rows` - número de filas del área

`columns` - número de columnas del área.

`doc` - modelo de contenedor de texto a usar.

Métodos

getDocument public Document getDocument()
Proporciona el modelo de documento asignado al área de texto.

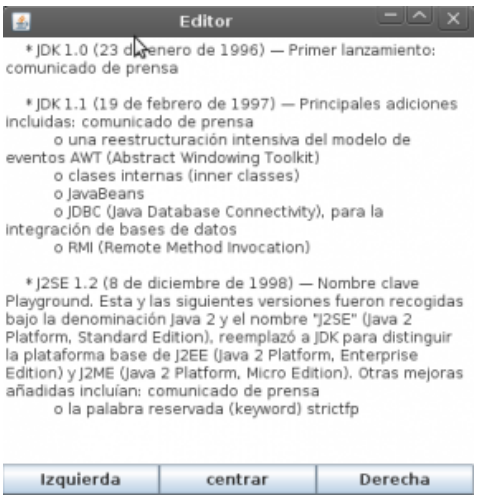


Figura 3.6: Ejemplo de una área de texto en java.

JFormattedTextField. Los campos de texto con formato permiten restringir las entradas del usuario. Por ejemplo, si se desea que el usuario solo tenga la posibilidad de introducir números enteros, se puede definir el siguiente campo de texto:

```
JFormattedTextField campoInt = new JFormattedTextField(
    NumberFormat.getIntegerInstance());
```

La clase `NumberFormat` cuenta además con los métodos estáticos

`getNumberInstance`

`getCurrencyInstance`

`getPercentInstance`

que proporcionan formateadores para números de coma flotante, valores monetarios y porcentajes, respectivamente. También existen formateadores para hora y fecha.

Constructores

JFormattedTextField La variantes son:

- public JFormattedTextField()
- public JFormattedTextField(Format format)

Parámetros:

`format` - formato que se desea usar.

Figura 3.7: Ejemplo de campos de texto con formato en java.

JComboBox. Proporciona un componente de selector múltiple. Se crea un ejemplar de la clase y después se añaden uno a uno los elementos con la función addItem la cual admite cualquier objeto. El selector múltiple llama al método `toString` de cada objeto para visualizarlo. Si se desea que los elementos sean editables, se llama al método `setEditable` con el argumento `true`.

```
JComboBox comboCaja = new JComboBox();
comboCaja.addItem("Elemento1");
comboCaja.addItem("Elemento2");
comboCaja.addItem("Elemento3");
...
```

Constructores

JComboBox Las variantes son:

- public JComboBox()
- public JComboBox(Object[] items)

Parámetros:

`items` - elementos que se desean agregar al selector múltiple.

Figura 3.8: Ejemplo de un selector múltiple en java.

JSlider. Los controles deslizantes permiten al usuario seleccionar una opción dentro de un espectro continuo de valores. Se puede especificar el valor mínimo, máximo e inicial. Además, se puede elegir si tendrá una posición horizontal o vertical y pueden ser adornados. A continuación se muestra el código para un control con marcas grandes rotuladas cada 20 unidades y marcas pequeñas cada 5, obligando al control deslizante a que salte hasta las marcas.

```
JSlider controlDeslizante =
  new JSlider(min, max, valorInicial);
controlDeslizante.setMajorTickSpacing(20);
controlDeslizante.setMinorTickSpacing(5);
controlDeslizante.setPaintTicks(true);
controlDeslizante.setSnapToTicks(true);
controlDeslizante.setPaintLabels(true);
```

Constructores

JSlider Las variantes son:

- public JSlider()
- public JSlider(int min, int max)
- public JSlider(int min, int max, int value)
- public JSlider(int orientation, int min, int max, int value)

Parámetros:

`orientation` - orientación del control.

`min` - valor mínimo.

`max` - valor máximo.

`value` - valor inicial.

Métodos

setMinorTickSpacing public void setMinorTickSpacing(int n)
Establece el espacio entre las marcas de menor tamaño.

Parámetros:

`n` - valor del espacio entre las marcas pequeñas.

setMajorTickSpacing public void setMajorTickSpacing(int n)
Establece el espacio entre las marcas de mayor tamaño.

Parámetros:

`n` - valor del espacio entre las marcas grandes.

setPaintLabels public void setPaintLabels(boolean b)
Establece si son visibles o no las etiquetas .

Parámetros:

`b` - las etiquetas son visibles si es `true` y no visibles si es `false`.

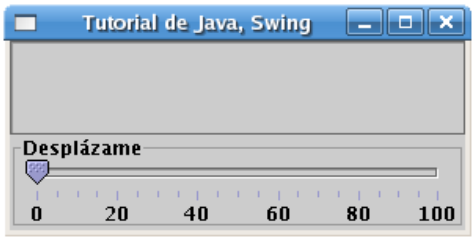


Figura 3.9: Ejemplo de un control deslizante en java.

JProgressBar. Una barra de progreso se compone de un rectángulo parcialmente lleno de un color y una cadena de la forma "*n* %" para indicar el progreso de cierto proceso. También se puede hacer uso de cadenas de texto para expresar el progreso de la barra.

```
barraDeProgreso = new JProgressBar(0,1000);
barraDeProgreso.setStringPainted(true);
...
if(barraDeProgreso > 900)
barraDeProgreso.setString("Casi hemos terminado");
```

Constructores

JProgressBar Las variantes son:

- public JProgressBar()
- public JProgressBar(int min,int max)

Parámetros:

`min` - valor mínimo de la barra.

`max` - valor máximo de la barra.

Métodos

setStringPainted public void setStringPainted(boolean b)
Establece la posibilidad de mostrar una cadena de texto describiendo el progreso.

Parámetros:

`b` - si es `true` se puede colocar una cadena de texto en lugar del valor porcentual y si es `false` se mantiene la configuración por omisión.

setString public void setString(String s)
Establece la cadena que se visualiza como descripción del progreso.

Parámetros:

`s` - cadena que se muestra como descripción del progreso.

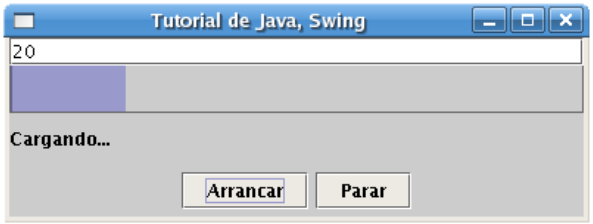


Figura 3.10: Ejemplo de una barra de progreso en java.

JTabbedPane. Las láminas con solapas son empleadas para fragmentar cuadros de diálogo complejos en subconjuntos de opciones relacionadas. Para crear una lámina con solapas, primero se construye un objeto de tipo JTabbedPane, y después se le van añadiendo solapas.

```
JTabbed laminaConSolapas = new JTabbedPane();
laminaConSolapas.addTab(titulo,icono,componente);
```

Constructores

JTabbedPane Las variantes son:

- public JTabbedPane()
- public JTabbedPane(int tabPlacement)

Parámetros:

`tabPlacement` - determina el lugar donde se colocaran la solapas alrededor de la lámina. Los valores definidos en la clase JTabbedPane son `TOP`, `BOTTOM`, `LEFT` y `RIGHT`.

Métodos

addTab Las variantes son:

- public void addTab(String title,Icon icon,Component component)
- public void addTab(String title,Component component)

Parámetros:

`title` - título del rotulo de la solapa.

`icon` - icono de la solapa.

`component` - componente que contendrá la lámina de la solapa.

Figura 3.11: Ejemplo de una lámina con solapas en java.

III.3.2 Eventos

Los componentes son fuentes de eventos, estos se deben transmitir a objetos oyentes de eventos. Las distintas fuentes pueden generar distintos tipos de eventos. Por ejemplo, un botón puede generar objetos `ActionEvent`, mientras que una ventana puede generar eventos `WindowEvent`. Los tipos de eventos de uso frecuente son:

ActionEvent
KeyEvent
AdjustementEvent
ItemEvent MouseWheelEvent
FocusEvent
WindowEvent

Un objeto oyente es un ejemplar de una clase que implementa una interfaz especial denominada interfaz de oyente. Cada interfaz tiene distintos métodos, los cuales deben ser definidos en su totalidad. Las interfaces siguientes están a la escucha de los eventos anteriores.

ActionListener
MouseMotionListener
AdjustmentListener
MouseWheelListener
FocusListener
WindowListener
ItemListener
WindowFocusListener
KeyListener
WindowStateListener
MouseListener

Por ejemplo, una clase que implemente la interfaz `ActionListener` se vería así

```
class MiOyente implements ActionListener
{
  ...
  public void actionPerformed(ActionEvent evento)
  {
    // aquí se pone la reacción al evento
    ...
  }
}
```

Los objetos de eventos cuentan con la función `getSource` que proporciona la fuente del evento.

Para registrar un objeto oyente en el objeto fuente se emplean unas líneas de código cuyo modelo es similar al siguiente

```
objetoFuente.addEventListener(objetoOyente);
```

Por ejemplo:

```
ActionListener oyente = ...;
JButton boton = new JButton("Ok");
boton.addActionListener(oyente);
```

En la tabla 3.3 se muestra un resumen del manejo de eventos.

III.3.3 Encargados de disposición

En Java los encargados de disposición establecen la forma en que se organizan los componentes en un contenedor. Es necesario escribir el código que se encarga de situar los componentes de la interfaz de usuario en los lugares que se desea que estén.

Para determinar el encargado de disposición para un contenedor se hace uso de la función

```
public void setLayout(LayoutManager mgr)
```

La cual recibe una referencia a un objeto que implemente la interfaz `LayoutManager`, como es el caso de todo encargado de disposición. De entre los posibles encargados con que se cuenta, están los siguientes.

FlowLayout. El encargado de disposición de flujo alinea los componentes horizontalmente hasta que no queda espacio, y después hace que empiece una nueva fila de componentes, manteniendo el tamaño de cada componente. De forma predeterminada los componentes se centran en el contenedor. Sin embargo, se puede modificar este comportamiento indicando en el constructor que alineación se desea, por ejemplo la alineación a la izquierda se logra con la constante `LEFT` como se ve a continuación:

```
panel.setLayout(new FlowLayout(FlowLayout.LEFT));
```

Tabla 3.3: Resumen del manejo de eventos.

Intefaz	Métodos	Parámetros	Eventos generados por
ActionListener	actionPerformed	ActionEvent	AbstractButton
			JComboBox
			JTextField
AdjustmentListener	adjustmentValueChanged	AdjustmentEvent	JScrollbar
ItemListener	itemStateChanged	ItemEvent	AbstractButtom
			JComboBox
FocusListener	focusGained	FocusEvent	Component
	focusLost		
KeyListener	keyPressed	KeyEvent	Component
	keyRelease		
	keyTyped		
MouseListener	mousePressed	MouseEvent	Component
	mouseReleased		
	mouseEntered		
	mouseExited		
	mouseClicked		
MouseMotionListener	mouseDragged	MouseEvent	Component
	mouseMoved		
MouseWheelListener	mouseWheelMoved	MouseWheelEvent	Component
WindowListener	windowClosing	WindowEvent	Window
	windowOpened		
	windowIconified		
	windowDeiconified		
	windowClosed		
	windowActivated		
	windowDeactivated		
WindowFocusListener	windowGainedFocus	WindowEvent	Window
	windowLostFocus		
WindowStateListener	windowStateChanged	WindowEvent	Window

Constructores

FlowLayout Las variantes son:

- public FlowLayout()
- public FlowLayout(int align)
- public FlowLayout(int align,int hgap,int vgap)

Parámetros:

`align` - valor de alineamiento. Los posibles valores de alineación son `CENTER`, `LEADING`, `LEFT`, `RIGHT` y `TRAILING`.

`hgap` - espacio horizontal entre los componentes.

`vgap` - espacio vertical entre los componentes.

BorderLayout. La disposición de borde divide el área de un contenedor en 5 zonas que son norte, oeste, centro, este y sur, lo cual permite seleccionar el lugar en que se desea colocar cada componente. Primero se sitúan los componentes de los bordes y luego el componente del centro ocupa el espacio restante, variando su tamaño si el contenedor es modificado en sus dimensiones. A diferencia de la disposición de flujo, hace que los componentes crezcan hasta llenar todo el espacio disponible. Los componentes se añaden especificando una constante llamada `CENTER`, `NORTH`, `SOUTH`, `EAST`, o `WEST`.

```
panel.setLayout(new BorderLayout());
panel.add(yellowButton, BorderLayout.SOUTH);
```

Constructores

BorderLayout Las variantes son:

- public BorderLayout()
- public BorderLayout(int hgap, int vgap)

Parámetros:

`hgap` - espacio horizontal entre los componentes.

`vgap` - espacio vertical entre los componentes.

GridLayout. La disposición de cuadrícula organiza todos los componentes en filas y columnas del mismo tamaño, distribuyéndose siempre en todo el contenedor. En el constructor se especifica el número de filas y columnas deseado. Los componentes se van añadiendo, empezando por la primera entrada de la primera fila, continuando de izquierda a derecha hasta llenar la fila y continuando con la siguiente de arriba hacia abajo.

```
panel.setLayout(new GridLayout(5, 4));
//Elemento columna 1, fila 1
```

```
panel.add(new JButton("1"));
//Elemento columna 2, fila 1
panel.add(new JButton("2"));
```

Constructores

GridLayout Las variantes son:

- public GridLayout()
- public GridLayout(int rows, int cols)
- public GridLayout(int rows, int cols, int hgap, int vgap)

Parámetros:

`rows` - número de filas de la cuadricula.

`cols` - número de columnas de la cuadricula.

`hgap` - espacio horizontal entre los componentes.

`vgap` - espacio vertical entre los componentes.

GridBagLayout. En una disposición de cuadrícula flexible, las filas y columnas pueden tener tamaños variables. Se pueden unir celdas contiguas para así contar con espacio suficiente para componentes más grandes. No es necesario que los componentes ocupen toda la celda y se puede especificar una alineación en cada una. A continuación se puede observar como se establece una disposición de cuadrícula flexible y se agrega un componente al contenedor.

```
panel.setLayout( new GridBagLayout());
panel.add( new JButton(),
           new GridBagConstraints(
                                0,
                                0,
                                1,
                                1,
                                30,
                                33,
                                GridBagConstraints.WEST,
                                GridBagConstraints.NONE,
                                new Insets(0, 16, 0, 0),
                                0,
                                0));
```

Se debe crear un objeto de tipo `GridBagConstraints` por cada componente agregado, cuyos argumentos especifican la forma en que estará dispuesto en la cuadrícula flexible. Se puede observar que en ningún momento se establece de forma explicita el numero de columnas y filas que conformaran la cuadrícula, en cambio el encargado

de disposición lo determinara según el número de componentes y su disposición a lo largo del contenedor. El constructor de la clase `GridBagConstraints` tiene los siguientes argumentos que inicializan los respectivos campos de la clase.

```
GridBagConstraints( int gridx,
                    int gridy,
                    int gridwidth,
                    int gridheight,
                    double weightx,
                    double weighty,
                    int anchor,
                    int fill,
                    Insets insets,
                    int ipadx,
                    int ipady)
```

Los argumentos `gridx` y `gridy` definen la posición del componente dento de la cuadrícula. El primero especifica la columna y el segundo la fila, siendo el punto de inicio la esquina superior izquierda y aumentando positivamente en dirección de izquierda a derecha para las columnas y de arriba hacia abajo para las filas. Con los argumentos `gridwidth` y `gridheight`, se establece el numero de columnas y filas que abarcara el componente respectivamente.

Los argumentos `weightx` y `weighty` se encargan de distribuir el espacio extra del contenedor entre las columnas y filas, respectivamente. Los valores varían entre 0 y 1, siendo el ultimo de mayor peso. El espacio se distribuye de forma proporcional al valor asignado, de esta forma, si se asigna un valor de cero a `weightx`, las columnas del componente no reciben espacio extra. Si todos existen componentes de distinta columna que tienen el mismo valor, mayor que cero, en el campo `weightx` entonces el espacio extra se repartirá en partes iguales entre sus columnas. En el caso de las filas es análogo. Estos valores afectan a toda la columna y la fila, no solo al espacio del componente, así que todos los componentes de la columna y fila deben tener los mismos valores o de otra forma el mayor valor es el que se toma en cuenta, descartando los demás.

Existen un conjunto de constantes definidas en la clase que pueden ser asignadas al campo `anchor`, el cual determina el lugar donde se colocara el componente cuando este es más pequeño que el área asignada. La orientación puede ser relativa a los componentes del contenedor, relativa a la línea base y absoluta como se muestra en la tabla 3.4.

El campo `fill` puede tomar cuatro posibles valores que permiten establecer si el componente llenara completamente el área signada, si se expande horizontalmente, verticalmente o se mantendrá en su tamaño original. Las constantes que determinan estos comportamientos son `BOTH`, `HORIZONTAL`, `VERTICAL`, y `NONE` respectivamente.

Tabla 3.4: Constantes de orientación para el campo anchor

Orientación absoluta	Relativa a los componentes	Relativa a la linea base
CENTER	PAGE_START	BASELINE
NORTH	PAGE_END	BASELINE_LEADING
NORTHEAST	LINE_START	BASELINE_TRAILING
EAST	LINE_END	ABOVE_BASELINE
SOUTH	FIRST_LINE_START	ABOVE_BASELINE_LEADING
SOUTHEAST	FIRST_LINE_END	ABOVE_BASELINE_TRAILING
SOUTHWEST	LAST_LINE_START	BELOW_BASELINE
WEST	LAST_LINE_END	BELOW_BASELINE_LEADING
NORTHWEST		BELOW_BASELINE_TRAILING

El argumento `insets` especifica la cantidad mínima de espacio entre el componente y los bordes. Es un objeto de tipo `Insets` cuyo constructor

```
Insets(int top, int left, int bottom, int right)
```

solicita como argumentos cuatro enteros que representan el espacio mínimo entre el componente y el borde de arriba, el borde izquierdo, el borde de abajo y el borde derecho, respectivamente.

Por ultimo, los campos `ipadx` y `ipady` especifican el ancho y la altura mínimos del componente en cuestión.

Para más información concerniente a Swing y AWT se recomienda consultar Horstmann and Cornell (2006a), Horstmann and Cornell (2006b) y Luis and Matilde (2001). Para cualquier duda sobre Java se puede acceder a la documentación completa en Oracle Corp. (2011).

III.4. Interfaz hombre-máquina

Las máquinas herramienta de CNC emplean dos teclados especiales, a saber:

Panel de control Para introducir instrucciones alfanuméricas, tales como programas o instrucciones manuales directas (IMDs). Como se aprecia en la figura 3.12, un panel de control es parecido al teclado de una computadora personal.

Panel de operación Para operar la máquina en modo manual o para controlar la ejecución de un programa. Como se puede observar en la figura 3.13, un panel de operación tiene una interfaz más orientada al operador de la máquina que al programador de la misma al ofrecer maneras de operar los ejes de la máquina y de ajustar las velocidades y otras propiedades del movimiento de la máquina, ya sea bajo el control de un humano (modo manual) o de la computadora del sistema de control numérico por computadora (modo automático).

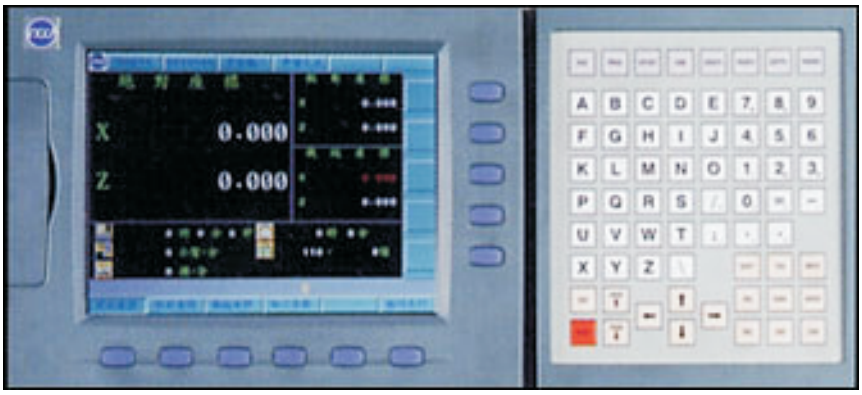


Figura 3.12: Panel de control de una máquina herramienta.

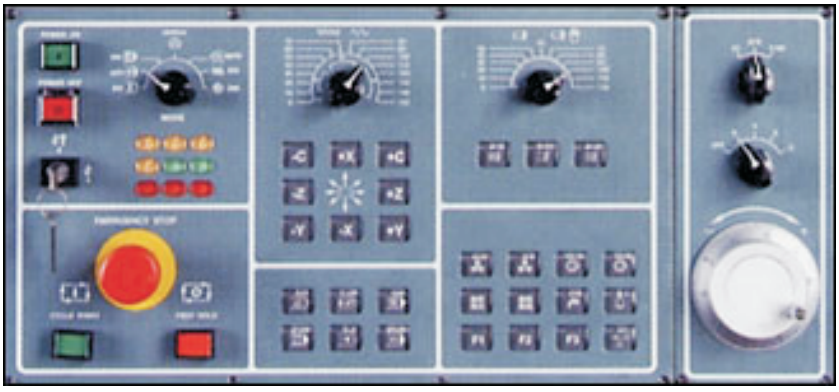


Figura 3.13: Panel de operación de una máquina herramienta.

Entre las funciones más comunes de un panel de operación se encuentran un paro de emergencia general, un botón para iniciar la ejecución de un programa y otro para pausarla, así como teclas etiquetadas con el nombre de un eje seguido de una dirección (e.g., X+ o X-) destinadas a dar instrucciones relativas a los ejes durante la operación manual. Asimismo, es común encontrar selectores de velocidades y un generador de pulsos.

III.5. Funciones que realiza un sistema de CNC

En ambos tipos de máquina herramienta de CNC, fresadora y torno, la elaboración de piezas se hace mediante la eliminación selectiva de material, ya sea rotando la herramienta de corte o la pieza, respectivamente. La remoción del material hecha por la máquina debe de ser determinada por el usuario; para ello se deben de especificar características de control especiales como son la velocidad de rotación del husillo y la velocidad de avance de la herramienta de corte.

La *velocidad del husillo* es lo que determina qué tan rápido debe girar la herramienta de corte, en la fresadora, o la pieza, en el torno, para hacer una desbastación de material adecuada; esto depende del tipo de material con el que se esté trabajando y la herramienta de corte. Existen otros parámetros relacionados con la velocidad del husillo tales como el sentido de rotación y las unidades de medida. Estos parámetros son controlados por otras funciones conocidas como funciones misceláneas que se mencionan más adelante.

La *velocidad de avance* de la herramienta de corte indica qué tan rápido es cortado el material, es decir, con qué velocidad avanza la herramienta de corte por la superficie a trabajar, en el caso de la fresadora, o la pieza, en el caso del torno. La velocidad de corte puede ser medida de dos maneras: unidad de medida por minuto o por revoluciones en donde las unidades de medida pueden ser en milímetros o pulgadas dependiendo del sistema métrico que se utilice.

La velocidad de rotación del husillo y la velocidad de avance de la herramienta de corte son parámetros que nos permiten cortar la superficie del material a trabajar para obtener la pieza deseada, es decir, estas velocidades determinan cómo ha de moverse la herramienta. Para describir instrucciones de maquinado completas falta saber cuál es la *trayectoria* que ha de seguir la herramienta; por eso otro factor importante en las máquinas herramienta es la existencia de un sistema coordenado. Los sistemas coordenados que se usan en las maquinas herramienta son el sistema coordenado rectangular y el sistema coordenado polar. Cuando definimos un sistema coordenado usamos ejes de referencia intersectados en un origen y tomamos este punto de intersección como el punto de referencia o punto cero; sin embargo existen otros ejes adicionales que se usan para otras tareas especificas, como rotaciones, donde se definen otros puntos de referencia. La manera en que se toman los ejes en las máquinas herramienta depende de la constitución (o arquitectura) cada una de las máquinas de CNC que se trabaje. Las máquinas herramienta usualmente tienen más de un eje coordenado de referencia. En el caso de la fresadora se tienen tres ejes, dos ejes horizontales, llamados eje X y eje Y, y un eje vertical, llamado eje Z y cuando existe un cuarto eje, éste generalmente es de rotación. En el torno el número de ejes se reduce en uno, pues hay un eje que siempre está fijo. Las figuras 3.15 y 3.14 muestran los ejes que generalmente se encuentran en este tipo de máquinas herramienta.

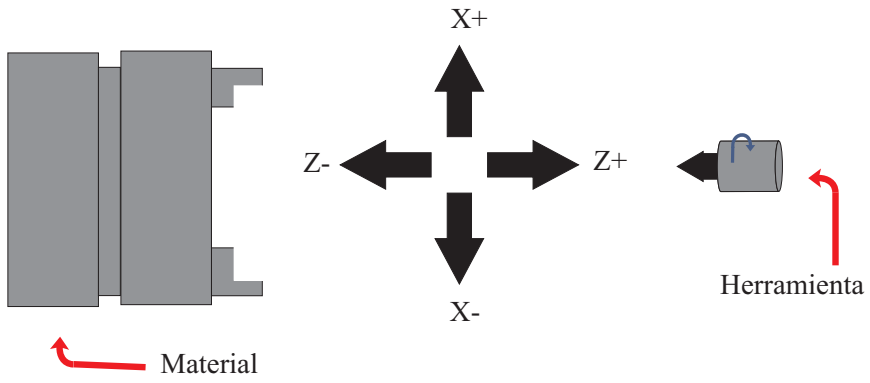


Figura 3.14: Ejes típicos de un torno.

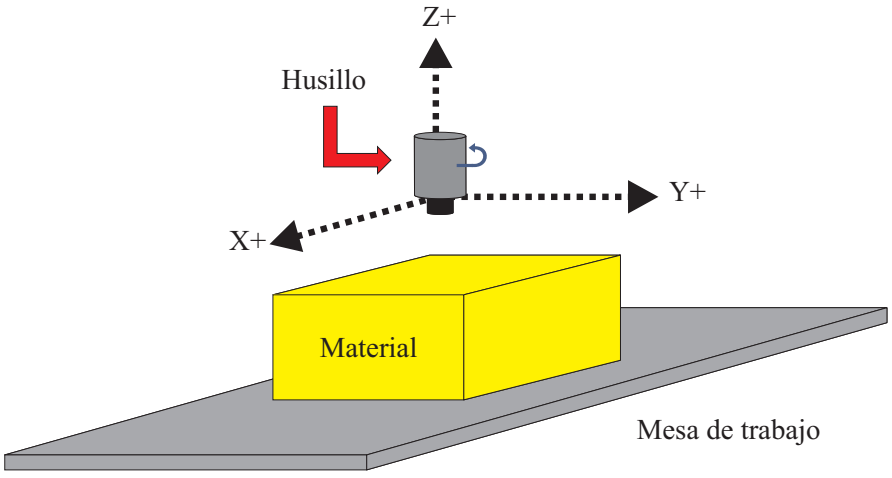


Figura 3.15: Ejes típicos de una fresadora.

Para este trabajo se consideran una fresadora vertical de tres ejes (el eje X es paralelo a la parte más larga de la mesa, el eje Y a la parte más angosta y el eje Z corresponde al desplazamiento vertical del husillo) y un torno horizontal de dos ejes; sin embargo, los resultados pueden ser extendidos a otras máquinas herramienta del mismo tipo sin dificultad.

III.6. Modos de operación de un sistema de CNC

A grandes rasgos, podemos decir que un sistema de CNC tiene dos modos de operación, a saber, el modo de operación manual y el modo de operación automático. El *modo de operación manual* se emplea para preparar una máquina herramienta antes de su operación automática y para maquinar piezas de forma manual. El *modo de operación automático* se emplea cuando se tiene un programa cargado en la memoria de la unidad de control del sistema de CNC y se desea ejecutarlo para producir alguna pieza.

El "modo de operación manual" en realidad está formado por los siguientes modos manuales de operación:

Inicio Sirve para ayudar a los ejes de la máquina a localizar su correspondiente punto de referencia (también conocido como cero), establecido usualmente con ayuda de uno o más interruptores de límite.

Rápido Se emplea para desplazar los ejes de la máquina a una velocidad de desplazamiento rápido (es decir, la velocidad empleada cuando se desea desplazar la herramienta sin realizar ningún corte). Esta velocidad de desplazamiento rápido usualmente se establece como una configuración del controlador.

Impuso También desplaza los ejes de la máquina pero a una velocidad acotada por la velocidad de impulso (en milímetros por minuto) establecida en el panel de operación.

Manivela Una manivela, en el caso de un sistema de CNC, es un generador de pulsos que permite seleccionar un eje a la vez junto con un multiplicador $\times 1$, $\times 10$ ó $\times 100$ del mínimo desplazamiento posible por parte del eje dado.

IMD Es posible ejecutar instrucciones que permitan cambiar el estado del controlador numérico para modificar parámetros tales como el sistema métrico a utilizar, o para realizar un cambio de herramienta por medio de instrucciones parecidas a las que se emplean para indicar conjuntos de pasos a realizar por la máquina en modo automático; este tipo de instrucciones se conocen como Instrucciones Manuales Directas.

Asimismo, el "modo de operación automático" en realidad requiere de los siguientes modos automáticos de operación:

Memoria Cuando el ejecutivo de programas está activo, lee un archivo de código G y va ejecutando las instrucciones contenidas en el mismo.

Editor Permite la modificación y creación de programas en código G para su posterior ejecución.

Administrador de archivos Usualmente un sistema de CNC puede trabajar con más de un programa, así que usualmente es necesario elegir un *programa activo* (el programa a editar o ejecutar) entre una colección de programas disponibles, así como copiar, borrar, transferir y realizar otras actividades de administración de archivos.

III.7. Modos manuales de operación

Los modos manuales de operación, salvo por el modo inicio, no son mutuamente excluyentes, así que no es inusual emplear más de uno de ellos al mismo tiempo, y es común que compartan la misma interfaz gráfica de usuario.

III.7.1 Modo inicio

Al encender la computadora encargada de las tareas de control de un sistema de CNC usualmente los registros que contienen la posición en que se encuentra cada uno de los ejes se inicializan a cero, a pesar de que nada garantiza que ésta será efectivamente la posición en que se encontrarán los ejes de la máquina herramienta. Por esta razón, para garantizar la operación correcta de la máquina, es necesario encontrar la verdadera posición de inicio de la máquina antes de intentar ejecutar cualquier clase de instrucción que dependa del posicionamiento correcto de los ejes de la máquina. El modo inicio fue creado con este propósito en mente, para permitir al usuario localizar la verdadera posición de inicio del sistema; usualmente el modo inicio es el modo en el que se encuentra el sistema de control al iniciar. La localización de la posición de inicio se realiza en la dirección indicada en el panel de operación por medio de la tecla de dirección de eje correspondiente (e.g., X+ inicia la búsqueda de la posición de inicio desplazando el eje X en la dirección positiva).

III.7.2 Modo rápido

A veces es necesario realizar un desplazamiento rápido de los ejes de la máquina para efectos de maquinado manual tales como el desplazamiento de la herramienta de corte para trabajar en otra parte de la pieza o para hacer un cambio herramienta. La velocidad de desplazamiento rápido es un valor configurado en la memoria del controlador, cuyo valor se puede variar ligeramente con opciones de porcentaje en el panel de operación (usualmente 25 %, 50 % y 100 %). El movimiento se realiza en la dirección elegida durante el tiempo que esté presionada la tecla de dirección de eje correspondiente (e.g., mientras se presione X+ el movimiento será en la dirección positiva del eje X).

III.7.3 Modo de avance constante

El modo de avance constante o modo impulso se emplea para desplazar los ejes de la máquina con un control de velocidad preciso, siendo éstas velocidades usualmente bajas, con el propósito de desbastar material durante la operación manual de la máquina. La velocidad de impulso se establece directamente por medio del panel de operación (aunque sería posible tener un enfoque parecido al de la velocidad de desplazamiento rápido almacenando un valor de referencia en la memoria del controlador y empleando una selección de porcentaje en el panel de operación). El movimiento se realiza en la dirección elegida durante el tiempo que esté presionada la tecla de dirección de eje correspondiente (e.g., mientras se presione X+ el movimiento será en la dirección positiva del eje X).

III.7.4 Modo manivela

Cuando resulta más relevante tener un control preciso de la posición de la herramienta durante un maquinado manual que tener un control preciso de la velocidad se emplea el modo manivela. El control de la posición se logra por medio de un generador de pulsos, donde cada pulso indica un incremento o decremento de la oposición del husillo.

III.7.5 Modo Introducción Manual de Datos (IMD)

El modo Introducción Manual de Datos (IMD) se emplea para introducir órdenes de configuración y de desplazamientos elaborados en los que se requiera controlar tanto la velocidad como el posicionamiento de manera precisa.

III.8. Modos automáticos de operación

La ejecución de un programa para un sistema de CNC (usualmente llamado programa de parte por contener las instrucciones de la parte a fabricar) se realiza en modo memoria, mientras que los modos editor y administrador de archivos en realidad sólo se usan para administrar los programas dentro del sistema.

III.8.1 Modo memoria

Este modo se emplea para ejecutar programas de control numérico por computadora que ya han sido creados previamente por medio del (modo) editor. Estos programas establecen las velocidades, posiciones, trayectorias y demás factores que se emplean para describir un conjunto de actividades para maquinar una parte sin la intervención directa del ser humano. Durante la ejecución de estos programas es posible modificar la velocidad de avance y otras opciones del mismo programa por medio del panel de operación.

III.8.2 Modo editor

Se emplea para editar y crear nuevos programas de parte que habrán de ser ejecutados de manera automática en modo memoria. Los editores integrados dentro de un sistema de CNC usualmente ofrecen características especiales, desde la simple numeración automática de cada línea (bloque) hasta la simulación del programa de parte.

III.8.3 Modo administrador de archivos

En el modo de administración de archivos es posible copiar programas desde un dispositivo externo, borrar programas para liberar memoria, copiar un programa existente en un nuevo archivo para crear (después de las modificaciones pertinentes) un programa parecido al original, etc., así como establecer un enlace remoto para CND; el CND consiste en leer las instrucciones de un programa desde un dispositivo externo por medio de un puerto de comunicaciones.

Para más información concerniente a los sistemas de CNC se recomienda consultar Smid (2003) y Koren (1983).

III.9. Condiciones de las pruebas

Para revelar la presencia de posibles defectos en la GUI al momento de integrarse con el resto de los módulos, se proponen pruebas de caja negra, las cuales se limitan a suministrar datos como entrada y estudiar la salida, sin importar lo que pueda estar haciendo

el módulo por dentro. Con estas pruebas se pretende validar el protocolo de comunicación y la interfaz entre el módulo de la GUI con el resto del sistema, al confirmar que los mensajes enviados y recibidos concuerdan.

Para confeccionar los casos de prueba de caja negra se proponen dos criterios:

Particiones de equivalencia. Este método intenta dividir el dominio de entrada de un programa en un número finito de clases de equivalencia. Se definen dos tipos de clases de equivalencia, las clases de equivalencia válidas, que representan entradas válidas al programa, y las clases de equivalencia no válidas, que representan valores de entrada erróneos.

Si una condición de entrada especifica un rango de valores, se tiene una clase valida (el rango) y dos no validas (los valores fuera del rango). Si son valores específicos se tiene una clase valida por valor. También se tienen clases de equivalencia no validas cuando se puede introducir valores no numéricos. Se asigna a cada clase de equivalencia un número único.

Se trata de escribir casos de prueba que cubran tantas clases válidas como sea posible y para cada clase no valida se crea un caso propio.

Análisis de valores límite. Complementa la técnica de partición de equivalencia de manera que se toman los casos de prueba donde se incluyen los valores límite en lugar de cualquier valor de los rangos.

En el presente trabajo se toman las siguientes consideraciones para los casos de prueba:

- Se usan milímetros como unidades de longitud para establecer las entradas en los casos de prueba, sin embargo, los casos con pulgadas son análogos.
- Para los controles gráficos (controles deslizantes, botones, botones de radio, etc.) no se determinan clases de equivalencia no validas, pues su naturaleza no permite estos casos.
- Las clases de equivalencia no validas de los campos editables se agrupan dentro de un mismo caso de prueba, pues al confirmar cambios, la validación trata por separado cada campo editado y no es posible que algún valor erróneo sea ignorado y aceptado.

Para más información concerniente a la ingeniería del software se recomienda consultar Sommerville (2005).

A continuación, se describen las clases de equivalencia derivadas de la GUI divididas en tablas por categoría de datos, además de las respectivas tablas de casos de prueba.

Tabla 3.5: Clases de equivalencia de la categoría operación de la máquina

Condición de entrada	Tipo	Clase de equivalencia válida
Modo de operación	Valor	1: Memoria (0x04, 0x00), 2: IMD (0x04, 0x01), 3: Administrador de archivos (0x04, 0x02), 4: Editor (0x04, 0x03), 5: Manivela (0x04, 0x04), 6: Avance constante (0x04, 0x05)

Tabla 3.6: Clases de equivalencia de la categoría movimiento de los ejes

Condición de entrada	Tipo	Clase de equivalencia válida
Retorno a cero	Valor	1: Eje X (0x09,0x01) 2: Eje Y (0x09,0x11) 3: Eje Z (0x09,0x21)
Corrección de la velocidad de avance rápido (%)	Rango	4: (0x08, corrección de la vel.) 0x00 $\leq$ corrección de la vel. $\leq$ 0x78
Velocidad de avance constante (mm/min)	Rango	5: (0x07, vel.cons.) 0 $\leq$ vel.cons. $\leq$ vel. máx. avance const.
Corrección de la velocidad de avance constante (%)	Rango	6: (0x06, corrección de la vel. cons.) 0 $\leq$ corrección de la vel. cons. $\leq$ 0x78
Movimiento constante del eje X positivo	Valor	7: ignorado (0x09,0x00) 8: seleccionado (0x09,0x01)
Movimiento constante del eje X negativo	Valor	9: ignorado (0x09,0x02) 10: seleccionado (0x09,0x03)
Movimiento constante del eje Y positivo	Valor	11: ignorado (0x09,0x10) 12: seleccionado (0x09,0x11)
Movimiento constante del eje Y negativo	Valor	13: ignorado (0x09,0x12) 14: seleccionado (0x09,0x13)
Movimiento constante del eje Z positivo	Valor	15: ignorado (0x09,0x20) 16: seleccionado (0x09,0x21)
Movimiento constante del eje Z negativo	Valor	17: ignorado (0x09,0x22) 18: seleccionado (0x09,0x23)
Factor de la manivela	Valor	19: Factor 1 (0x0D,0x02) 20: Factor 10 (0x0D,0x03) 21: Factor 100 (0x0D,0x04)
Selección del eje para la manivela	Valor	22: Eje X (0x0D,0x05) 23: Eje Y (0x0D,0x06) 24: Eje Z (0x0D,0x07)
Corrección de la velocidad de avance de corte (%)	Rango	25: (0x06, vel. de corte) 0x00 $\leq$ vel. de corte $\leq$ 0x78

Tabla 3.7: Clases de equivalencia de la categoría movimiento de los ejes (Cont.)

Condición de entrada	Tipo	Clase de equivalencia válida	Clase de equivalencia no válida
Velocidad de avance de corte por omisión (mm/min)	Rango	26: (0x34, v.a.c.o.) 0x00 $\leq$ v.a.c.o. v.a.c.o. $\leq$ v.máx.a.c.o.	27: Un valor no numérico 28: v.a.c.o. $<$ 0x00 29: v.a.c.o. $>$ v.máx.a.c.o.
Velocidad máxima de avance de corte (mm/min)	Rango	30: (0x35, v.máx.a.c.o.) v.a.c.o. $\leq$ v.máx.a.c.o. v.máx.a.c.o. $\leq$ 0x10C6	31: Un valor no numérico 32: v.máx.a.c.o. $<$ v.a.c.o. 33: v.máx.a.c.o. $>$ 0x10C6
Velocidad de avance constante por omisión (mm/min)	Rango	34: (0x07, v.a.const.o.) 0x00 $\leq$ v.a.const.o. v.a.const.o. $\leq$ v.máx.a.const.	35: Un valor no numérico 36: v.a.const.o. $<$ 0x00 37: v.a.const.o. $>$ v.máx.a.const.
Velocidad máxima de avance constante (mm/min)	Rango	38: (0x37, v.máx.a.const.) v.a.const.o. $\leq$ v.máx.a.const. v.máx.a.const. $\leq$ 0x10C6	39: Un valor no numérico 40: v.máx.a.const. $<$ v.a.const.o. 41: v.máx.a.const. $>$ 0x10C6
Velocidad de avance (mm/min)	Rango	42: (0x38, v.a.) 0x00 $\leq$ v.a. $\leq$ v.a.máx.	43: Un valor no numérico 44: v.a. $<$ 0x00 45: v.a. $>$ v.a.máx.
Velocidad máxima de avance (mm/min)	Rango	46: (0x3C, v.a.máx.) v.a. $\leq$ v.a.máx. $\leq$ 0x10C6	47: Un valor no numérico 48: v.a.máx. $<$ v.a. 49: v.a.máx. $>$ 0x10C6
Velocidad de avance de retorno a cero (mm/min)	Rango	50: (0x3A, v.a.r.c.) 0x00 $\leq$ v.a.r.c. $\leq$ 0x10C6	51: Un valor no numérico 52: v.a.r.c. $<$ 0x00 53: v.a.r.c. $>$ 0x10C6
Rango de desplazamiento del eje X (mm)	Rango	54: (0x3D, r.d.ejeX) 0x00 $\leq$ r.d.ejeX $\leq$ 0x863	55: Un valor no numérico 56: r.d.ejeX $<$ 0x00 57: r.d.ejeX $>$ 0x863
Rango de desplazamiento del eje Y (mm)	Rango	58: (0x3E, r.d.ejeY) 0x00 $\leq$ r.d.ejeY $\leq$ 0x863	59: Un valor no numérico 60: r.d.ejeY $<$ 0x00 61: r.d.ejeY $>$ 0x863
Rango de desplazamiento del eje Z (mm)	Rango	62: (0x3F, r.d.ejeZ) 0x00 $\leq$ r.d.ejeZ $\leq$ 0x863	63:Un valor no numérico 64: r.d.ejeZ $<$ 0x00 65: r.d.ejeZ $>$ 0x863
Referencia de dirección de retorno del eje X	Valor	66: Negativo (0x40, 0x00) 67: Positivo (0x40, 0x01)	
Referencia de dirección de retorno del eje Y	Valor	68: Negativo (0x41, 0x00) 69: Positivo (0x41, 0x01)	
Referencia de dirección de retorno del eje Z	Valor	70: Negativo (0x42, 0x00) 71: Positivo (0x42, 0x01)	

Tabla 3.8: Clases de equivalencia de la categoría husillo

Condición de entrada	Tipo	Clase de equivalencia válida	Clase de equivalencia no válida
Dirección	Valor	1: Manecillas (0x0A, 0x00) 2: En contra (0x0A, 0x01)	
Movimiento	Valor	3: Rotando (0x0A, 0x02) 4: Detenido (0x0A, 0x03)	
Velocidad (rpm)	Rango	5: (0x0B, velocidad) 0x00 $\leq$ velocidad $\leq$ 0xBB8	
Corrección de la velocidad (%)	Rango	6: (0x0C, corrección) 0x32 $\leq$ corrección $\leq$ 0x78	
Velocidad mínima (rpm)	Rango	7: (0x43, vel.mín.) 0x00 $\leq$ vel.mín. $\leq$ 0xFFFF	8: Un valor no numérico 9: vel.mín. $<$ 0x00 10: vel.mín. $>$ 0xFFFF
Velocidad máxima (rpm)	Rango	11: (0x44, vel.máx.) 0x00 $\leq$ vel.máx. $\leq$ 0xFFFF	12: Un valor no numérico 13: vel.máx. $<$ 0x00 14: vel.máx. $>$ 0xFFFF
Corrección de velocidad mínima (%)	Rango	15: (0x45, c.vel.mín.) 0x00 $\leq$ c.vel.mín. $\leq$ 0xFF	16: Un valor no numérico 17: c.vel.mín. $<$ 0x00 18: c.vel.mín. $>$ 0xFF
Corrección de velocidad máxima (%)	Rango	19: (0x46, c.vel.máx) 0x00 $\leq$ c.vel.máx $\leq$ 0xFF	20: Un valor no numérico 21: c.vel.máx $<$ 0x00 22: c.vel.máx $>$ 0xFF

Tabla 3.9: Clases de equivalencia de la categoría dispositivos

Condición de entrada	Tipo	Clase de equivalencia válida
Dispositivos	Valor	1: Luz desactivada (0x0F, 0x00) 2: Luz activada (0x0F, 0x01) 3: Luz automática (0x0F, 0x02) 4: Refrigerante desactivado (0x0F, 0x10) 5: Refrigerante activado (0x0F, 0x11) 6: Refrigerante automático (0x0F, 0x12)

Tabla 3.10: Clases de equivalencia de la categoría planificación de ruta

Condición de entrada	Tipo	Clase de equivalencia válida	Clase de equivalencia no válida
Sistema métrico por omisión	Valor	1: Internacional (0x4E, 0x00) 2: Anglosajón (0x4E, 0x01)	
Modo de dimensionamiento por omisión	Valor	3: Absoluto (0x50, 0x00) 4: Incremental (0x50, 0x01)	
Desplazamiento del área de trabajo (mm)	Rango	5: (0x51, índice del área, eje, valor) 0x00 $\leq$ índice del área $\leq$ 0x06 eje: x = 0x00, y = 0x01, z = 0x02 -0x863 < valor < 0x863	6: Un valor no numérico (x,y,z) 7: valor < -0x863 8: valor > 0x863
Compensación por herramienta de corte (mm)	Rango	9: (0x52, índice de herr., ld, value) 0x00 $\leq$ índice de herramienta $\leq$ 0x09 ld: long. = 0x00, diam. = 0x01 -0x863 < value < 0x863	10: Un valor no numérico (l,d) 11: value < -0x863 12: value > 0x863
Área de trabajo activa	Rango	13: (0x19, índice del área) 0x00 $\leq$ índice del área < 0x07	

Tabla 3.11: Clases de equivalencia de la categoría soporte de la herramienta

Condición de entrada	Tipo	Clase de equivalencia válida	Clase de equivalencia no válida
Diámetro máximo de la herramienta (mm)	Rango	1: (0x4B, diám.máx.) 0x00 $\leq$ diám.máx. $\leq$ 0x10C6	2: Un valor no numérico 3: diám.máx. < 0x00 4: diám.máx. > 0x10C6
Longitud máxima de la herramienta (mm)	Rango	5: (0x4C, long.máx.) 0x00 $\leq$ long.máx. $\leq$ 0x10C6	6: Un valor no numérico 7: long.máx. < 0x00 8: long.máx. > 0x10C6

Tabla 3.12: Clases de equivalencia de la categoría cambio automático de herramienta

Condición de entrada	Tipo	Clase de equivalencia válida	Clase de equivalencia no válida
Capacidad	Rango	1: (0x47, capacidad) 0x00 $\leq$ capacidad $\leq$ 0xFF	2: Un valor no numérico 3: capacidad < 0x00 4: capacidad > 0xFF
Índice máximo	Rango	5: (0x48, índice máx.) 0x00 $\leq$ índice máx. $\leq$ capacidad	6: Un valor no numérico 7: índice máx. < 0x00 8: índice máx. > capacidad
Tipo de selección	Valor	9: Fijo (0x49, 0x00) 10: Aleatorio (0x49, 0x01)	
Herramienta activa	Rango	11: (0x1A, índice de herramienta) 0x00 $\leq$ índice de herramienta índice de herramienta $\leq$ índice máx.	12: Un valor no entero 13: índice de herr. < 0x00 14: índice de herr. > índice máx.

Tabla 3.13: Clases de equivalencia del teclado

Condición de entrada	Tipo	Clase de equivalencia válida
Teclas del alfabeto	Rango	1: A - Z (0x01,valor) 0x41≤valor ≤0x5A
Teclas numéricas	Rango	2: 0 - 9 (0x01,valor) 0x30≤valor ≤0x39
Otras teclas de caracteres	Valor	3: “ ‘ ” (0x01,0xC0), 4: “ - ” (0x01,2D), 5: “ = ” (0x01,0x3D), 6: “ [” (0x01,0x5B), 7: ”] ” (0x01,0x5D), 8: “ ; ” (0x01, 0x3B), 9: ” ’ ” (0x01,0xDE), 10: “ , ” (0x01,0x2C), 11: “ . ” (0x01,0x2E), 12: “ / ” (0x01,0x2F)
Cursores	Valor	13: Arriba (0x01,0x26), 14: Abajo (0x01,0x28), 15: Izquierda (0x01,0x25), 16: Derecha (0x01,0x27)
Otras	Valor	17: Retroceso (0x01,0x08), 18: Tabulación (0x01,0x09), 19: Entrar (0x01,0x0A), 20: Bloq Mayús (0x01,0x14), 21: Shift (0x01,0x10), 22: Control (0x01,0x11), 23: Alt (0x01,0x12), 24: Página arriba (0x01,0x21), 25: Página abajo (0x01,0x22)

Tabla 3.14: Clases de equivalencia de las tareas de edición de archivos

Condición de entrada	Tipo	Clase de equivalencia válida
Administrador y editor de archivos	Valor	1: Copiar 0x00 2: Abrir 0x01 3: Guardar 0x02 4: Guardar como 0x03 5: Actualizar 0x04 6: Eliminar 0x05 7: Cambiar nombre 0x06

Tabla 3.15: Casos de prueba de la categoría operación de la máquna

Nº de caso	1	2	3	4	5	6
Clase de equivalencia	1	2	3	4	5	6
Modo de operación	(0x04,0x00)	(0x04,0x01)	(0x04,0x02)	(0x04,0x03)	(0x04,0x04)	(0x04,0x05)

Tabla 3.16: Casos de prueba de la categoría movimiento de los ejes

Nº de caso	1	2	3	4	5
Clases de equivalencia	1, 4a, 5a, 6a, 19, 22	2, 4b, 5b, 6b, 20, 23, 25a	3, 21, 24, 25b	7, 11, 15 8, 12, 16	9, 13, 17 10, 14, 18
Retorno a cero	(0x09,0x01)	(0x09,0x11)	(0x09,0x21)		
Corrección de la vel. de avance rápido	(0x08,0x00)	(0x08,0x78)			
Velocidad de avance constante	(0x07,0x00)	(0x07,0xC8)			
Corrección de la vel. de avance constante	(0x06,0x00)	(0x06,0x78)			
Movimiento constante del eje X				(0x09,0x00)* (0x09,0x01)*	(0x09,0x02)* (0x09,0x03)*
Movimiento constante del eje Y				(0x09,0x10)* (0x09,0x11)*	(0x09,0x12)* (0x09,0x13)*
Movimiento constante del eje Z				(0x09,0x20)* (0x09,0x21)*	(0x09,0x22)* (0x09,0x23)*
Factor de la manivela	(0x0D,0x02)	(0x0D,0x03)	(0x0D,0x04)		
Selección del eje de la manivela	(0x0D,0x05)	(0x0D,0x06)	(0x0D,0x07)		
Corrección de la vel. de avance de corte		(0x06,0x00)	(0x06,0x78)		

*Los dos valores se encuentran relacionados por el comportamiento del botón: presionar-soltar

Tabla 3.17: Casos de prueba de la categoría movimiento de los ejes (Cont.)

Nº de caso	6	7	8	9
Clases de equivalencia	26a, 30a, 34a, 38a, 42a, 46a, 50a	30b, 26b , 34b, 38b, 42b, 46b, 50b	54a, 58a, 62a, 66, 68, 70	54b, 58b, 62b, 67, 69, 71
Velocidad de avance de corte por omisión	(0x34,0x00)	(0x34,0x10C6)		
Velocidad máxima de avance de corte	(0x35,0x00)	(0x35,0x10C6)		
Velocidad de avance constante por omisión	(0x07,0x00)	(0x07,0x10C6)		
Velocidad máxima de avance constante	(0x37,0x00)	(0x37,0x10C6)		
Velocidad de avance	(0x38,0x00)	(0x38,0x10C6)		
Velocidad máxima de avance	(0x3C,0x00)	(0x3C,0x10C6)		
Velocidad de avance de retorno a cero	(0x3A,0x00)	(0x3A,0x10C6)		
Rango de desplazamiento del eje X			(0x3D,0x00)	(0x3D,0x863)
Rango de desplazamiento del eje Y			(0x3E,0x00)	(0x3E,0x863)
Rango de desplazamiento del eje Z			(0x3F,0x00)	(0x3F,0x863)
Referencia de la dirección de retorno del eje X			(0x40, 0x00)	(0x40, 0x01)
Referencia de dirección de retorno de eje Y			(0x41, 0x00)	(0x41, 0x01)
Referencia de dirección de retorno del eje Z			(0x42, 0x00)	(0x42, 0x01)

Tabla 3.18: Casos de prueba de la categoría movimiento de los ejes (Cont.)

Nº de caso	10	11	12
Clase de equivalencia	27, 31, 35, 39, 43, 47, 51, 55, 59, 63	28, 32, 36, 40, 44, 48, 52, 56, 60, 64	29, 33, 37, 41, 45, 49, 53, 57, 61, 65
Velocidad de avance de corte por omisión	“number”	(0x34,-0x01)	(0x34,0x10C7)
Velocidad máxima de avance de corte	“number”	(0x35,-0x01)	(0x35,0x10C7)
Velocidad de avance constante por omisión	“number”	(0x07,-0x01)	(0x07,0x10C7)
Velocidad máxima de avance constante	“number”	(0x37,-0x01)	(0x37,0x10C7)
Velocidad de avance	“number”	(0x38,-0x01)	(0x38,0x10C7)
Velocidad máxima de avance	“number”	(0x3C,-0x01)	(0x3C,0x10C7)
Velocidad de avance de retorno a cero	“number”	(0x3A,-0x01)	(0x3A,0x10C7)
Rango de desplazamiento del eje X	“number”	(0x3D,-0x01)	(0x3D,0x864)
Rango de desplazamiento del eje Y	“number”	(0x3E,-0x01)	(0x3E,0x864)
Rango de desplazamiento del eje Z	“number”	(0x3F,-0x01)	(0x3F,0x864)

Tabla 3.19: Casos de prueba de la categoría husillo

Nº de caso	1	2	3	4
Clase de equivalencia	1, 3, 5a, 6a	2, 4, 5b, 6b	7a, 11a, 15a, 19a	7b, 11b, 15b, 19b
Dirección	(0x0A, 0x00)	(0x0A, 0x01)		
Movimiento	(0x0A, 0x02)	(0x0A, 0x03)		
Velocidad	(0x0B,0x00)	(0x0B,0xBB8)		
Corrección de la velocidad	(0x0C,0x32)	(0x0C,0x78)		
Velocidad mínima			(0x43,0x00)	(0x43,0xFFFF)
Velocidad máxima			(0x44,0x00)	(0x44,0xFFFF)
Corrección de vel. mínima			(0x45,0x00)	(0x45,0xFF)
Corrección de vel. máxima			(0x46,0x00)	(0x46,0xFF)

Tabla 3.20: Casos de prueba de la categoría husillo (Cont.)

Nº de caso	5	6	7
Clase de equivalencia	8, 12, 16, 20	9, 13, 17, 21	10, 14, 18, 22
Dirección			
Movimiento			
Velocidad			
Corrección de la velocidad			
Velocidad mínima	“number”	(0x43,-0x01)	(0x43,0x10000)
Velocidad máxima	“number”	(0x44,-0x01)	(0x44,0x10000)
Corrección de velocidad mínima	“number”	(0x45,-0x01)	(0x45,0x100)
Corrección de velocidad máxima	“number”	(0x46,-0x01)	(0x46,0x100)

Tabla 3.21: Casos de prueba de la categoría dispositivos

Nº de caso	1	2	3
Clase de equivalencia	1, 4	2, 5	3, 6
Luz	(0x0F, 0x00)	(0x0F, 0x01)	(0x0F, 0x02)
Refrigerante	(0x0F, 0x10)	(0x0F, 0x11)	(0x0F, 0x12)

Tabla 3.22: Casos de prueba de la categoría planificación de ruta

Nº de caso	1	2	3
Clase de equivalencia	1, 3, 5a, 9a, 13a, 14a	2, 4, 5b, 9b, 13b, 14b	6, 10
Sistema métrico por omisión	(0x4E, 0x00)	(0x4E, 0x01)	
Modo de dim. por omisión	(0x50, 0x00)	(0x50, 0x01)	
Desplazamiento del área de trabajo	(0x51,índice,eje,-0x863)	(0x51,índice,eje,0x863)	“number”
Compensación por herr. de corte	(0x52,índice,ld,-0x863)	(0x52,índice,ld,0x863)	“number”
Área de trabajo activa	(0x19,0x00)	(0x19,0x06)	

Tabla 3.23: Casos de prueba de la categoría planificación de ruta (Cont.)

Nº de caso	4	5
Clase de equivalencia	7, 11	8, 12
Sistema métrico por omisión		
Modo de dimensionamiento por omisión		
Desplazamiento del área de trabajo	(0x51,índice,eje,-0x864)	(0x51,índice,eje,0x864)
Compensación por herramienta de corte	(0x52,índice,ld,-0x864)	(0x52,índice,ld,0x864)
Área de trabajo activa		

Tabla 3.24: Casos de prueba de la categoría soporte de la herramienta

Nº de caso	1	2	3	4	5
Clase de equivalencia	1a, 5a	1b, 5b	2, 6	3, 7	4, 8
Diam. máx. de la herr.	(0x4B,0x00)	(0x4B,0x10C6)	“number”	(0x4B,-0x01)	(0x4B,0x10C7)
Long. máx. de la herr.	(0x4C,0x00)	(0x4C,0x10C6)	“number”	(0x4C,-0x01)	(0x4C,0x10C7)

Tabla 3.25: Casos de prueba de la categoría cambio automático de herramienta

Nº de caso	1	2	3	4	5
Clase de equivalencia	1a, 5a, 9, 11a	1b, 5b, 10, 11b	2, 6, 12	3, 7, 13	4, 8, 14
Capacidad	(0x47,0x00)	(0x47,0xFF)	“number”	(0x47,-0x01)	(0x47,0x100)
Índice máximo	(0x48,0x00)	(0x48,0xFF)	“number”	(0x48,-0x01)	(0x48,0x100)
Tipo de selección	(0x49,0x00)	(0x49, 0x01)			
Herramienta activa	(0x1A,0x00)	(0x1A,0xFF)	“number”	(0x1A,-0x01)	(0x1A,0x100)

Tabla 3.26: Casos de prueba del teclado

Nº de caso	1	2	3	4	5
Clase de equivalencia	1a,3,13,17	1b,4,14,18	2a,5,15,19	2b,6,16,20	7,21
Teclas del alfabeto	(0x01,0x41)	(0x01,0x5A)			
Teclas numéricas			(0x01,0x30)	(0x01,0x39)	
Otras teclas	(0x01,0xC0),	(0x01,2D),	(0x01,0x3D),	(0x01,0x5B),	(0x01,0x5D)
Cursores	(0x01,0x26)	(0x01,0x28)	(0x01,0x25)	(0x01,0x27)	
Otras	(0x01,0x08)	(0x01,0x09)	(0x01,0x0A)	(0x01,0x14)	(0x01,0x10)

Tabla 3.27: Casos de prueba del teclado (Cont.)

Nº de caso	6	7	8	9
Clase de equivalencia	8,22	9,23	10,24	11,12,25
Teclas del alfabeto				
Teclas numéricas				
Otras teclas	(0x01, 0x3B),	(0x01,0xDE),	(0x01,0x2C),	(0x01,0x2E),(0x01,0x2F)
Cursores				
Otros	(0x01,0x11)	(0x01,0x12)	(0x01,0x21)	(0x01,0x22)

Tabla 3.28: Casos de prueba de las tareas de edición de archivos

Nº de caso	1	2	3	4	5	6	7
Clase de equivalencia	1	2	3	4	5	6	7
Administrador y editor de archivos	0x00	0x01	0x02	0x03	0x04	0x05	0x06

IV. RESULTADOS

IV.1. Clase NewComp

public class NewComp

La clase `NewComp` proporciona métodos para generar componentes gráficos configurados, con el objetivo de solo tener que establecer las opciones necesarias para la apariencia y comportamiento del componente, encapsulando el código mayormente repetitivo para estas tareas. También se pueden encontrar clases internas auxiliares para obtener cierto comportamiento especial de algunos componentes, así como métodos para conversiones de unidades y para el acceso a la información almacenada en archivos de propiedades. Los archivos de propiedades son:

- FontOptions.properties - En este archivo se almacenan las opciones del tamaño de fuente de la GUI. Se almacenan en parejas formadas por un nombre clave y el valor que representan, p. ej. *default_titleButton=4*.
- StringsGui.properties - En este archivo se almacenan todas las cadenas de texto que aparecen en la GUI. Se almacenan en parejas formadas por un nombre clave y la cadena de texto correspondiente, p. ej. *rapid=Rapido*. Si se quisiera traducir la aplicación a otro idioma distinto al español, se debe crear otro archivo con los mismos nombres clave pero con las cadenas en el idioma deseado.

IV.1.1 Métodos públicos

changeFont public static void changeFont(String fontId)

Cambia la cadena inicial de los nombres claves de los tamaños de fuente, con el fin de usar los mismos nombres clave en el mismo archivo, pero con distinto valor. Esto permite que se pueda variar el tamaño de fuente cuando se cambia de modo de operación, si se tienen menos componentes en pantalla se puede aumentar el tamaño de la fuente lo que provoca componentes mas grandes y se aprovecha el espacio de sobra. También se pueden establecer tamaños de fuente para diferentes resoluciones de pantalla. Los nombres claves para las fuentes se almacenan en la forma `option_codeName`, donde `codeName` es el nombre clave que se usa para referirse en el código a un tamaño de fuente. La cadena al inicio `option` es la que se puede elegir con este método.

Parámetros:

`fontId` - El identificador del grupo de fuentes que se quiere usar.

getButton Las variantes son:

- public static JButton getButton(String iconName, ActionListener listener)

- public static JButton getButton(String iconName, MouseListener listener)
- public static JButton getButton(String text, String sizeFont, String color, ActionListener listener)
- public static JButton getButton(String text, String iconName, String sizeFont, String color, ActionListener listener)

Proporciona un botón que puede o no tener icono y/o rótulo, además establece un oyente de eventos.

Parámetros:

`text` - Nombre clave que identifica a una cadena de texto que se agrega al rótulo del componente.

`iconName` - El nombre clave con que se relaciona al icono.[1]

`sizeFont` - Nombre clave del tamaño de fuente.

`color` - Nombre del color de la fuente como se establece en el lenguaje HTML.

`listener` - El oyente de eventos para el botón. Puede ser un oyente `ActionListener` para eventos del botón o de tipo `MouseListener` para estar al pendiente de los eventos de los botones del ratón.

getComboBox public static JComboBox getComboBox(String sizeFont)
Proporciona un selector múltiple.

Parámetros:

`sizeFont` - Nombre clave del tamaño de fuente para el texto de la lista.

getFTF Las variantes son:

- public static JFormattedTextField getFTF(boolean isSi, String sizeFont)
- public static JFormattedTextField getFTF(String whole, String sizeFont)

El primer método proporciona un campo de texto formateado para visualizar cantidades que estén expresadas en nanómetros a milímetros o pulgadas dependiendo del sistema métrico establecido. El segundo proporciona un campo de texto formateado para cantidades enteras. No son editables ni pueden obtener el foco[2].

Parámetros:

`isSi` - Establece el sistema métrico internacional si es verdadero o ingles si es falso.

`whole` - Establece el numero máximo de cifras visibles en campos enteros.

`sizeFont` - Nombre clave del tamaño de fuente.

[1] Véase Resources.java
[2] Véase las clases internas NumberFormatter e IntFormatter

getIcon public static ImageIcon getIcon(String key)
Proporciona un icono.

Parámetros:

`iconName` - El nombre clave con que se relaciona al icono[3].

getIntegerTxtField public static JFormattedTextField getIntegerTxtField(String sizeFont)
Proporciona un campo de texto con formato numérico entero y editable.

Parámetros:

`sizeFont` - Nombre clave del tamaño de fuente.

getLabel La variantes son:

- public static JLabel getLabel()
- public static JLabel getLabel(String text, String sizeFont)
- public static JLabel getLabel(String text, String sizeFont, String color)

Proporciona una etiqueta.

Parámetros:

`text` - Nombre clave que identifica a una cadena de texto que se agrega al rótulo del componente.

`sizeFont` - Nombre clave del tamaño de fuente.

`color` - Nombre del color de la fuente como se establece en el lenguaje HTML.

getLabelIcon public static JLabel getLabelIcon(String iconName)
Proporciona una etiqueta con un icono y sin texto.

Parámetros:

`iconName` - El nombre clave con que se relaciona al icono.

getNumberTxtField public static JFormattedTextField getNumberTxtField(String sizeFont)
Proporciona un campo de texto con formato numérico de coma flotante y editable.

Parámetros:

`sizeFont` - Nombre clave del tamaño de fuente.

getPanel Las variantes son:

- public static JPanel getPanel(LayoutManager layout)
- public static JPanel getPanel(String text, String sizeFont, LayoutManager layout)

[3] Véase Resources.java

- public static JPanel getPanel(String text_1, String text_2, String sizeFont, LayoutManager layout)

Proporciona una lámina con borde, borde y rótulo o con borde y rótulo especial.

Parámetros:

`text text_1` - Nombre clave que identifica a una cadena de texto que se agrega al rótulo del componente.

`text_2` - Texto extra para agregar al rótulo además del texto correspondiente al nombre clave.

`sizeFont` - Nombre clave del tamaño de fuente.

`layout` - Establece el encargado de disposición de este contenedor.

getProgressBar public static JProgressBar getProgressBar(String sizeFont)
Proporciona una barra de progreso con porcentaje.

Parámetros:

`sizeFont` - Nombre clave del tamaño de fuente.

getRadioButton public static JRadioButton getRadioButton(String text, String sizeFont, String color, boolean isSelected, ActionListener listener)
Proporciona un botón de radio.

Parámetros:

`text` - Nombre clave que identifica a una cadena de texto que se agrega al rótulo del componente.

`sizeFont` - Nombre clave del tamaño de fuente.

`color` - Nombre del color de la fuente como se establece en el lenguaje HTML.

`isSelected` - Establece como seleccionado o no al botón de radio.

`listener` - El oyente de eventos para el botón.

getSlider Las variantes son:

- public static JSlider getSlider(int min, int max, int value)
- public static JSlider getSlider(int minorTick, int majorTick, int min, int max, int value)

El primer método proporciona un control deslizante con marcas inteligentes que se adaptan a los valores mínimo y máximo. Para esto, se calcula el máximo común divisor del mínimo y máximo para usarlo como distancia entre las marcas. El segundo proporciona un control deslizante donde se tiene que especificar la distancia entre las marcas.

Parámetros:

`minorTick` - Espacio entre las marcas pequeñas.

`majorTick` - Espacio entre las marcas grandes.

`min` - Valor mínimo del control.

`max` - Valor máximo del control.

`value` - Valor inicial.

getString public static String getString(String text)
Proporciona una cadena de texto que corresponde al nombre clave.

Parámetros:

`text` - Nombre clave que identifica a una cadena de texto.

getStringWithFormat public static String getStringWithFormat(String text, String sizeFont, String color)
Proporciona una cadena de texto con formato HTML.

Parámetros:

`text` - Nombre clave que identifica a una cadena de texto que se agrega al rótulo del componente.

`sizeFont` - Nombre clave del tamaño de fuente.

`color` - Nombre del color de la fuente como se establece en el lenguaje HTML.

getTable public static JTable getTable(String[] name, Integer[] size, String[] date, String sizeFont)
Proporciona una tabla específicamente para visualizar un directorio. [4]

Parámetros:

`name` - Lista de los nombres de los archivos.

`size` - Lista de los tamaños de los archivos.

`date` - Lista de las fechas de modificación de los archivos.

`sizeFont` - Nombre clave del tamaño de fuente.

getTableModel public static TableModel getTableModel(String[] name, Integer[] size, String[] date)
Proporciona un modelo de tabla específicamente para visualizar un directorio.

Parámetros:

`name` - Lista de los nombres de los archivos.

`size` - Lista de los tamaños de los archivos.

`date` - Lista de las fechas de modificación de los archivos.

[4] Véase la clase interna BrowserTable

getTextArea public static JTextArea getTextArea(int rows, int columns, String sizeFont, boolean bold)

Proporciona un área de texto con un modelo de documento plano (`PlainDocument`).

Parámetros:

`rows` - Establece el numero de filas del área.

`columns` - Establece el numero de columnas del área.

`sizeFont` - Nombre clave del tamaño de fuente.

`bold` - Establece la fuente como negrita cuando es verdadero y como plano cuando es falso.

getTxtField public static JTextField getTxtField(String text, String sizeFont)

Proporciona un campo de texto que no es editable ni puede tener el foco.

Parámetros:

`text` - Nombre clave que identifica a una cadena de texto.

`sizeFont` - Nombre clave del tamaño de fuente.

getTxtFieldEditable public static JTextField getTxtFieldEditable(String sizeFont)

Proporciona un campo de texto editable.

Parámetros:

`sizeFont` - Nombre clave del tamaño de fuente.

inToNm public static int inToNm(float in)

Realiza la conversión de pulgadas a nanómetros.

Parámetros:

`in` - Cantidad de pulgadas a convertir

mmToNm public static int mmToNm(float mm)

Realiza la conversión de milímetros a nanómetros.

Parámetros:

`mm` - Cantidad de milímetros a convertir

nmToIn public static float nmToIn(int nm)

Realiza la conversión de nanómetros a pulgadas.

Parámetros:

`nm` - Cantidad de nanómetros a convertir.

nmToMm public static float nmToMm(int nm)

Realiza la conversión de nanómetros a milímetros.

Parámetros:

`nm` - Cantidad de nanómetros a convertir.

IV.1.2 Métodos privados

mcd private static int mcd(int b, int c)
Proporciona el máximo común divisor de dos números de forma recursiva.

Parámetros:

`b` - Un número entero.

`c` - Un número entero.

getIntSizeFont private static int getIntSizeFont(String sizeFont)
Proporciona el valor entero correspondiente al nombre clave del tamaño de fuente.

Parámetros:

`sizeFont` - Nombre clave del tamaño de fuente.

getStringFontSize private static String getStringFontSize(String fontSize)
Proporciona el valor como cadena de texto correspondiente al nombre clave del tamaño de fuente.

Parámetros:

`sizeFont` - Nombre clave del tamaño de fuente.

IV.1.3 Campos privados

fontOption private static String fontOption
Este campo almacena el nombre clave que distingue un grupo de fuentes de otro.[5]

fontBundle private static ResourceBundle fontBundle
Es un manojo de recursos en el cual se cargan los tamaños de fuente definidos en el archivo FontOptions.properties. Estos números se cargan en forma de cadena de texto, se necesita de un paso extra para poder usarlos como enteros.[6]

stringBundle private static ResourceBundle stringBundle
Es un manojo de recursos en el cual se cargan las cadenas de texto definidas en el archivo StringsGui.properties.

IV.1.4 Clases internas

BrowserTable private static class BrowserTable extends AbstractTableModel
La clase interna BrowserTable es un modelo de tabla específicamente para visualizar el contenido de un directorio. Extiende a la clase abstracta AbstractTableModel de la cual es necesario implementar solo tres métodos y además se invalida uno.

Constructor y métodos

BrowserTable private BrowserTable(String[] name_,Integer[] size_,String[] date_)
El constructor inicializa los tres campos privados.

[5] Véase el método changeFont.
[6] Véase el método getIntSizeFont.

getRowCount public int getRowCount()
Propuesto por AbstractTableModel, proporciona el número total de filas de la tabla.

getColumnCount public int getColumnCount()
Propuesto por AbstractTableModel, proporciona el número total de columnas de la tabla. En este caso es constante igual a tres.

getValueAt public Object getValueAt(int row, int column)
Propuesto por AbstractTableModel, proporciona el elemento de la fila `row` y columna `column` de la tabla.

getColumnName public String getColumnName(int c)[7]
Proporciona el nombre correspondiente a la columna `c`.

getBrowserTable public static BrowserTable getBrowserTable(String[] name_, Integer[] size_, String[] date_)
Proporciona un ejemplar del modelo BrowserTable.

Campos

name private String[] name
Arreglo para almacenar los nombres de los archivos.

date private String[] date
Arreglo para almacenar la fecha de modificación de los archivos.

size private Integer[] size
Arreglo para almacenar el tamaño de los archivos.

NumberFormatter private static class NumberFormatter extends DefaultFormatter
La clase interna NumberFormatter proporciona un formato para campos de texto. Recibe cantidades en nanómetros y lo visualizan en milímetros o pulgadas. Extiende a la clase abstracta DefaultFormatter.

Constructor y métodos

NumberFormatter private NumberFormatter(boolean isSi)
Crea un objeto de tipo NumberFormatter y establece el formato internacional o el anglosajón.

valueToString public String valueToString(Object value)
Originalmente de la case DefaultFormatter, transforma el valor del campo en la cadena que se visualiza en el campo. Verifica que el valor `value` sea de tipo `Integer`, sino lanza una excepción. El valor entero se supone esta en nanómetros, por lo cual se realiza una conversión a las unidades correspondientes al sistema métrico establecido. Teniendo la conversión se sigue a darle el formato correspondiente, el cual, en el caso del sistema internacional es un número de hasta cuatro cifras enteras y tres decimales. Mientras que en el sistema anglosajón se trata de un número de hasta tres cifras enteras y cuatro decimales.

[7] Invalidado de AbstractTableModel

stringToValue public Object stringToValue(String text)
Originalmente de la case DefaultFormatter, analiza el texto en el campo y lo transforma en un objeto. El objeto en este caso es de tipo `Float`.

getFormat public static NumberFormatter getFormat(boolean isSi)
Proporciona un ejemplar del formateador NumberFormatter.

Campos

isSi_ private boolean isSi_
Almacena la opción del sistema métrico. Si se establece como verdadero entonces es el sistema internacional, en caso de que se establezca como falso entonces se usa el sistema anglosajón.

IntFormatter private static class IntFormatter extends DefaultFormatter
La clase IntFormatter proporciona un formato para campos de texto. Visualiza cantidades enteras con un limite de cifras. Extiende a la clase abstracta DefaultFormatter.

Constructor y métodos

IntFormatter private IntFormatter(int whole)
Crea un objeto de tipo IntFormatter y establece el numero máximo de cifras que se visualizaran.

valueToString public String valueToString(Object value)
Originalmente de la case DefaultFormatter, transforma el valor del campo en la cadena que se visualiza en el campo. Verifica que el valor `value` sea de tipo `Integer`, sino lanza una excepción. El valor se transforma en cadena de texto y se rellenan las posiciones sobrantes con ceros. Las cantidades con mas cifras de las permitidas se anulan.

stringToValue public Object stringToValue(String text)
Originalmente de la case DefaultFormatter, analiza el texto en el campo y lo transforma en un objeto. El objeto en este caso es de tipo `Integer`.

getFormat public static IntFormatter getFormat(int whole)
Proporciona un ejemplar de IntFormatter.

Campos

whole_ private int whole_
Almacena el número máximo de cifras que puede visualizar el componente.

IV.2. Clase Resources

public class Resources extends ListResourceBundle

La clase Resources administra los iconos usados en toda la GUI. A cada icono se le asigna un nombre clave con el que es llamado. La clase extiende a ListResourceBundle la cual permite poner todos los recursos en una matriz de objetos, después se encarga de efectuar

la búsqueda. Por ejemplo, para obtener el icono asociado a la clave “ok” se crea un objeto de tipo Resource y se emplea el método `handleGetObject` heredado de ListResourceBundle de esta forma

```
Resources res = new Resources();
ImageIcon icon = (ImageIcon) res.handleGetObject("ok");
```

Se realiza una refundición al tipo de objeto esperado.

IV.2.1 Constructor

Resources public Resources()
Crea un nuevo objeto de tipo Resources.

IV.2.2 Métodos protegidos

getContents protected Object[][] getContents()[8]
Proporciona un arreglo matricial de objetos que contiene los recursos.

IV.2.3 Campos privados

iconsPath private static final String iconsPath
Ruta de acceso a los iconos de la GUI.

iconsKPath private static final String iconsKPath
Ruta de acceso a los iconos del teclado.

contents private static final Object[][] contents
Arreglo matricial de recursos. Esta formado por parejas de objetos de tipo `String` y `ImageIcon`. El primer elemento de la pareja es el nombre clave con que se solicitara el recurso y el segundo es un icono.

IV.3. Clase StartUIManager

public class StartUIManager

La clase StartUIManager se encarga de establecer el tema visual del entorno gráfico llamado Nimbus.

IV.3.1 Métodos públicos

StartLookAndFeell public static void StartLookAndFeell()
Primero obtiene un arreglo con información de todos lo temas visuales instalados en el sistema y busca el tema Nimbus, el cual si es encontrado, lo establece. Este método debe ser llamado antes de crear cualquier componente de la interfaz gráfica, sin excepciones.

[8] Propuesto por ListResourceBundle

IV.4. Clase Keyboard

public class Keyboard extends JPanel

La clase Keyboard crea un teclado virtual, figura 4.1. Para generar eventos de teclado se recurre a la clase Robot, la cual originalmente tiene el propósito de hacer pruebas automáticas de un programa. La clase Robot cuenta con las funciones `keyPress`, para generar el evento nativo de presionar una tecla, y `keyRelease`, para generar el evento nativo de soltar una tecla antes presionada. Entonces, si se desea presionar virtualmente una tecla, digamos la tecla correspondiente a la letra "C", el código es el siguiente:

```
Robot robot = new Robot();
robot.keyPress(KeyEvent.VK_C);
robot.keyRelease(KeyEvent.VK_C);
```

El argumento pasado a las funciones son valores constantes definidos en la clase KeyEvent.

La clase extiende a JPanel pues la idea es que al crear un objeto de esta, es como si se obtuviera un teclado que se puede empotrar donde se desee, pues todas las filas de botones se van agregando directamente sobre la lámina.

Es importante que en la configuración del teclado del sistema se establezca la distribución norteamericana para que funcione correctamente.

Figura 4.1: Teclado

IV.4.1 Constructor

Keyboard public Keyboard()

Crea un objeto de tipo Keyboard. Inicializa el Robot, el cual puede generar una excepción si la configuración de la plataforma no permite el control de entradas de bajo nivel. También los campos privados a sus valores iniciales.

Se usa un arreglo de cadenas de texto para describir el teclado. Cada fila del arreglo corresponde a una fila del teclado como se muestra en la figura 4.2. Cada tecla tiene un nombre clave con el que se hace referencia en el arreglo. Se solicita la creación de cada fila y se agrega a la lámina principal.

IV.4.2 Métodos privados

createRow private JPanel createRow(String[] keyID)

Proporciona una lámina con las teclas solicitadas.

Parámetros:

Figura 4.2: Filas del teclado

`keyID` - Arreglo de nombres claves de teclas virtuales.

createLastRow private JPanel createLastRow(String[] keyID_1, String[] keyID_2, String[] keyID_3)

Proporciona una lámina con las teclas solicitadas. El método simplemente es el caso especial de la ultima fila de un teclado físico correspondiente a las filas 6, 7 y 8 del arreglo, la fila donde se encuentran la tecla de espacio, teclas modificadoras, teclas cursor, entre otras. Se puede prescindir de este método y escribir el código directamente en el constructor.

Parámetros:

`keyID_1 keyID_2 keyID_3` - Arreglos de nombres claves de teclas virtuales.

getKey private JButton getKey(final String vkID)

Proporciona un botón que hace de tecla virtual. A cada botón se le agrega un oyente de acciones donde se generan los eventos virtuales de presionar y soltar la tecla cuando se presiona el botón. Contiene varios casos pues las teclas modificadoras tienen un comportamiento especial conocido como "*teclas pegajosas*". Para lograr este comportamiento cada botón especial cuanta con una variable de tipo `boolean`, la cual informa si la tecla fue presionada o soltada, `true` o `false` respectivamente. Así realiza la tarea respectiva de presionar o soltar la tecla al volver a presionar el botón.

Parámetros:

`vkID` - Nombre clave que identifica una tecla.

createControlButtons private void createControlButtons()

Inicializa los dos botones "Control".

createShiftButtons private void createShiftButtons()

Inicializa los dos botones "Shift".

IV.4.3 Campos privados

firstControl private boolean firstControl

Tiene la función de indicar si el primer botón "Control" ha sido agregado al teclado.

firstShift private boolean firstShift
Tiene la función de indicar si el primer botón "Shift" ha sido agregado al teclado.

control private JButton[] control
Botones que simulan teclas "Control". Es necesario que sean campos de objeto por que se debe tener acceso a los dos dentro de su oyente de acciones para cambiar el icono cada vez que son presionados.

shift private JButton[] shift
Botones que simulan teclas "Shift". Es necesario que sean campos de objeto por que se debe tener acceso a los dos dentro de su oyente de acciones para cambiar el icono cada vez que son presionados.

shiftKeypress private boolean shiftKeypress
Indica si las teclas "Shift" se encuentran presionadas o no, `true` o `false` respectivamente.

altKeypress private boolean altKeypress
Indica si la tecla "Alt" se encuentra presionada o no, `true` o `false` respectivamente.

altGrKeypress private boolean altGrKeypress
Indica si la tecla "AltGr" se encuentra presionada o no, `true` o `false` respectivamente.

controlKeypress private boolean controlKeypress
Indica si la tecla "Control" se encuentra presionada o no, `true` o `false` respectivamente.

keyTask private Robot keyTask
Ejemplar de tipo Robot para crear eventos de teclas virtuales.

idValue private static KeyIdValue idValue
Lista de nombres clave y valores de las teclas virtuales.[9]

IV.4.4 Clases internas

KeyIdValue private static class KeyIdValue extends ListResourceBundle
La clase KeyIdValue es una lista donde se relacionan los nombres en clave de las teclas con su respectivo valor de tecla virtual. La clase extiende a ListResourceBundle de la cual se hereda el método `hadleGetObject`, encargado de realizar la búsqueda del elemento deseado.

Constructor y métodos

KeyIdValue public KeyIdValue()
Crea un nuevo objeto de tipo KeyIdValue.

[9] Véase la clase interna KeyIdValue.

getContents public Object[][] getContents()
Proporciona el arreglo matricial que contiene la relación de nombres clave y valores de teclas virtuales.

Campos

contents private static final Object[][] contents
Arreglo matricial constante que contiene las relación de nombres clave y valores de teclas virtuales.

IV.5. Clase DataCategory

public abstract class DataCategory implements DataConst

La clase DataCategory sirve como base para las distintas categorías de datos proporcionando un modelo a seguir y algunos métodos de utilidad.

IV.5.1 Métodos públicos

initData Las variantes son:

- public abstract boolean initData(int id,int value)
- public abstract boolean initData(int id,long value)

Se propone este método para que en el se inicialicen los datos de cada categoría. A cada dato le corresponde un identificador, ya sea de la clase KbdCmdId o DataConst, entonces en un `switch` se administra la información recibida. Implementa la interfaz DataConst para acceder directamente a las constantes.

Parámetros:

`id` - Identificador del dato, definido en KbdCmdId o DataConst. Si el identificador coincide con algún valor dentro del método, se retorna verdadero. El valor es falso en otro caso.

`value` - El valor que se asignara al dato correspondiente.

getView public abstract Component getView(Id viewID)
Se propone este método para proporcionar los componentes destinados para el marco superior, los cuales comúnmente solo muestran información y no son editables. Por lo general, se trata de una lámina que contiene un grupo de componentes que se relacionan según el tipo de información que muestran.

Parámetros:

`viewID` - Identificador del componente o grupo de componentes que se quiere obtener. Se trata de un elemento de la enumeración Id contenida en la clase DataConst.

getEditor public abstract Component getEditor(Id editorID)

Se propone este método para proporcionar los componentes destinados para el marco inferior, los cuales comúnmente serán botones, controles o campos editables. Estos componentes deberán contar con oyentes de eventos donde informen de los cambios realizados en ellos. [10]

Parámetros:

`editorID` - Identificador del componente que se quiere obtener. Se trata de un elemento de la enumeración Id contenida en la clase DataConst.

getDataCategoryID public abstract String getDataCategoryID()

Proporciona una cadena de texto que identifica la categoría.

getValue public abstract String getValue(Id dataID)

Se propone este método para proporciona el valor del dato correspondiente al identificador. El valor se retorna en forma de cadena de texto.

Parámetros:

`dataID` - Identifcador correspondiente al dato que se desea consultar.

setCommandValue Las variantes son:

- public boolean setCommandValue(int id,int value)
- public boolean setCommandValue(int id,long value)

Se propone este método para que en el se establezcan los nuevos valores generados por los controles del teclado, tanto en los campos como en los componentes correspondientes. A cada dato le corresponde un identificador de la clase KbdCmdId, entonces en un `switch` se administra la información recibida.

Parámetros:

`id` - Identificador del dato, definido en KbdCmdId. Si el identificador coincide con algún valor dentro del método, se retorna verdadero. El valor es falso en otro caso.

`value` - El valor que se asignara al dato correspondiente.

setDataValue Las variantes son:

- public abstract boolean setDataValue(int id,int value)
- public abstract boolean setDataValue(int id,long value)

Se propone este método para que en el se establezcan los nuevos valores generados por tareas ajenas al teclado, tanto en los campos como en los componentes correspondientes. A cada dato le corresponde un identificador de la clase DataConst, entonces en un `switch` se administra la información recibida.

Parámetros:

[10] Véase la clase InfoManager para ver los métodos con los que cuenta para informar de cambios al código nativo.

`id` - Identificador del dato, definido en DataConst. Si el identificador coincide con algún valor dentro del método, se retorna verdadero. El valor es falso en otro caso.

`value` - El valor que se asignara al dato correspondiente.

applyChange public abstract void applyChange()
Se propone este método para confirmar cambios en componentes editables. Cuando se edita un componente se debe agregar su identificador a la lista `changed`, lo cual se logra agregando un oyente de eventos que realice esta tarea. La lista se consulta al llamar a este método para saber que valores sufrieron cambios, entonces, los campos y componentes correspondientes toman el nuevo valor y se informa al código nativo. Este método se puede colocar en un oyente de acciones de un botón para que al presionarlo se confirmen los cambios.

cancelChange public abstract void cancelChange()
Se propone este método para cancelar cambios realizados en componentes editables. Cuando se edita un componente se debe agregar su identificador a la lista `changed`, la cual se consulta al llamar a este método para saber que valores sufrieron cambios. El componente editable vuelve a su valor anterior sin alterar otro componentes. Este método se puede colocar en un oyente de acciones de un botón para que al presionarlo se cancelen los cambios.

toNm public int toNm(String metricSystem,float value) Proporciona la conversión en nanómetros de una magnitud. Dependiendo del sistema métrico, realiza las operaciones correctas para pasar milímetros o pulgadas a nanómetros.

Parámetros:

`metricSystem` - Identificador del sistema métrico en uso. Si se trata del sistema internacional (“international”) se asume que la magnitud se encuentra en milímetros. En el caso del sistema anglosajón (“english”) la magnitud se asume en pulgadas.

`value` - Magnitud de la cual se desea conocer el valor equivalente en nanómetros.

toMmOrIn public float toMmOrIn(String metricSystem,int value) Proporciona la conversión en milímetros o pulgadas de una magnitud en nanómetros, dependiendo del sistema métrico.

Parámetros:

`metricSystem` - Identificador del sistema métrico en uso. Si se trata del sistema internacional (“international”) la magnitud se cambia a milímetros. En el caso del sistema anglosajón (“english”) la magnitud se cambia a pulgadas.

`value` - Magnitud en nanómetros de la cual se desea conocer el valor equivalente en milímetros o pulgadas.

IV.5.2 Campos protegidos

changed protected ArrayList <Id>changed

Lista de identificadores que corresponden a componentes editables. Si un componente sufre cambios, se debe agregar a la lista su identificador con el objetivo de tener un control al confirmar o cancelar dichos cambios de edición. Estas tareas deben ser atendidas en los métodos `applyChange` y `cancelChange` respectivamente.

IV.5.3 Campos constantes

MAX_UINT_32 public static final long MAX_UINT_32

Campo constante que almacena el valor máximo que puede tomar un unsigned int de 32 bits en C++.

MAX_UINT_16 public static final int MAX_UINT_16

Campo constante que almacena el valor máximo que puede tomar un unsigned int de 16 bits en C++.

MAX_UINT_8 public static final short MAX_UINT_8

Campo constante que almacena el valor máximo que puede tomar un unsigned int de 8 bits en C++.

IV.6. Clase MachineOperation

public class MachineOperation extends DataCategory

La clase MachineOperation agrupa los datos que describen el modo y estado de operación de la máquina. Extiende a la clase DataCategory.

IV.6.1 Constructor

MachineOperation public MachineOperation() Crea un objeto de tipo MachineOperation que representa una categoría de datos de la máquina.

IV.6.2 Métodos públicos

initData Las variantes son:

- public abstract boolean initData(int id,int value)
- public abstract boolean initData(int id,long value)

Propuesto por DataCategory, inicializa los campos del paro de emergencia y modo de operación.

getView public Component getView(Id viewID)

Propuesto por DataCategory, proporciona los componentes que visualizan el estado del paro de emergencia y el modo de operación. Véase la tabla 4.1.

getEditor public Component getEditor(Id editorID)

Propuesto por DataCategory, la categoría no cuenta con componentes editables.

Tabla 4.1: Componentes que visualizan datos de la categoría operación de la máquina

Datos	Identificador	Figura
Paro de emergencia	Id.EMERGENCY_STOP	4.3a
Modo de operación	Id.OPERATION_MODE	4.3b

(a) Paro de emergencia

(b) Modo de operación

Figura 4.3: Componentes visualizadores del modo de operación y estado del paro de emergencia de la máquina

getDataCategoryID public String getDataCategoryID()
Propuesto por DataCategory, proporciona la cadena "machineOperation" como identificador de la categoría.

getValue public String getValue(Id dataID) Propuesto por DataCategory, proporciona una cadena con el nombre del modo de operación en actual.

setCommandValue Las variantes son:

- public boolean setCommandValue(int id,int value)
- public boolean setCommandValue(int id,long value)

Propuesto por DataCategory, establece los nuevos valores del paro de emergencia y modo de operación.

setDataValue Las variantes son:

- public abstract boolean setDataValue(int id,int value)
- public abstract boolean setDataValue(int id,long value)

Propuesto por DataCategory, la categoría no lo usa.

applyChange public void applyChange()
Propuesto por DataCategory, la categoría no lo usa.

cancelChange public void cancelChange()
Propuesto por DataCategory, la categoría no lo usa.

getOperationMode public static String getOperationMode()
Proporciona el nombre del modo de operación actual.

IV.6.3 Campos privados

emergencyStopOpt private static final String[] emergencyStopOpt
Campo constante formado por dos cadenas de texto, las cuales son nombres clave de los iconos correspondientes a los dos posibles estados que puede presentar el paro de emergencia. Las cadenas son: “stop” y “continue”.

operationModeOpt private static final String[] operationModeOpt
Campo constante formado por ocho cadenas de texto, las cuales son nombres clave que identifican los nombre de los modos de operación. Las cadenas son: “memoryMode”, “mdiMode”, “fileManagerMode”, “editorMode”, “handwheelMode”, “jogMode”, “rapidMode” y “zeroReturnMode”.

emergencyStop private static Integer emergencyStop
Almacena el valor correspondiente del estado del paro de emergencia. Si es igual a cero es que se encuentra detenido y si es igual a uno es que continua.

operationMode private static Integer operationMode
Almacena el valor correspondiente al modo de operación en uso. El valor varía de 0 a 7 y corresponde al modo de operación en el orden que aparecen en el campo constante `operationModeOpt`.

oprtnModeLabel private static JLabel oprtnModeLabel
Etiqueta donde se visualiza el modo de operación actual.

emergencyStopIcon private static JLabel emergencyStopIcon
Etiqueta donde se visualiza el icono correspondiente al estado del paro de emergencia.

IV.7. Clase PathPlanning

public class PathPlanning extends DataCategory

La clase PathPlanning agrupa los datos que afectan la planificación de la ruta de corte. Extiende a la clase DataCategory.

IV.7.1 Constructor

PathPlanning public PathPlanning()
Crea un objeto de tipo PathPlanning que representa una categoría de datos de la máquina.

IV.7.2 Métodos públicos

initData Las variantes son:

- public boolean initData(int id, int value)
- public boolean initData(int id, long value)

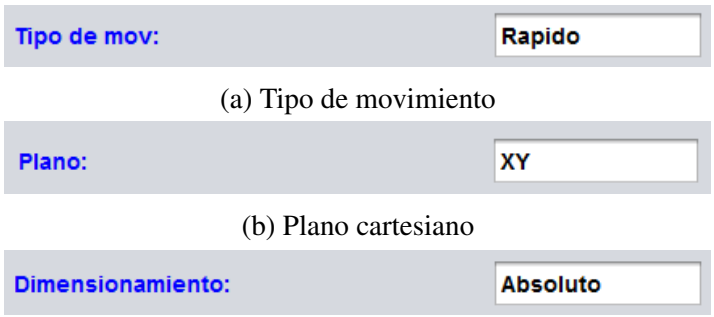
Inicializa los campos de tipo de movimiento, plano cartesiano, sistema métrico por omisión, sistema métrico, modo de dimensionamiento, modo de dimensionamiento por omisión, valor de los ejes x, y, z de la posición destino, el índice de la compensación de trabajo activo, el índice de la compensación de la herramienta activa y el número total de compensaciones del área de trabajo.

getView public Component getView(Id viewID)

Propuesto por DataCategory, proporciona los componentes que muestran información respecto al tipo de movimiento, plano cartesiano, modo de dimensionamiento, sistema métrico, cero pieza y posición destino. Véase la tabla 4.2.

Tabla 4.2: Componentes que visualizan datos de la categoría planificación de ruta

Datos	Identificador	Figura
Tipo de movimiento	Id.MOTION_TYPE	4.4a
Plano cartesiano	Id.CARTESIAN_PLANE	4.4b
Modo de dimensionamiento	Id.DIMENSIONING_MODE	4.4c
Cero pieza	Id.PROGRAM_ZERO	4.4e
Posición destino	Id.ATP	4.4d



(a) Tipo de movimiento

(b) Plano cartesiano

(c) Modo de dimensionamiento

(d) Posición destino

(e) Cero pieza

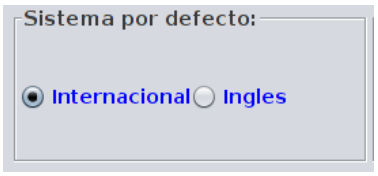
Figura 4.4: Componentes visualizadores de los datos de la categoría planificación de ruta

getEditor public Component getEditor(Id editorID)

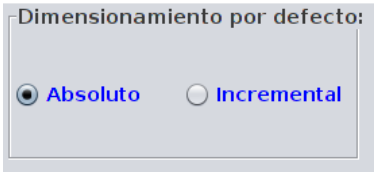
Propuesto por DataCategory, proporciona componentes que permiten elegir el sistema métrico y modo de dimensionamiento por omisión, la compensación de trabajo y de la herramienta. Los dos últimos permiten editar sus valores también. Véase la tabla 4.3.

Tabla 4.3: Componentes editables de la categoría planificación de ruta

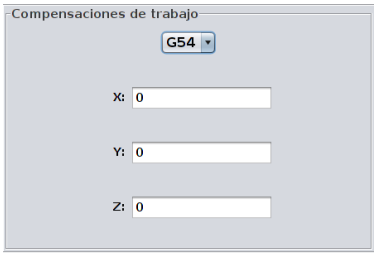
Datos	Identificador	Figura
Sistema métrico por omisión	Id.DEFAULT_METRIC_SYSTEM	4.5a
Modo de dimensionamiento por omisión	Id.DEFAULT_DIMENSIONING_MODE	4.5b
Compensaciones del área de trabajo	Id.WORK_OFFSET	4.5c
Compensaciones de las herramientas	Id.TOOL_OFFSET	4.5d



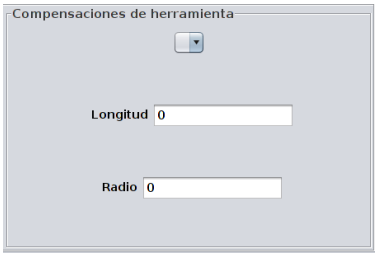
(a) Selección del sistema métrico por omisión.



(b) Selección del modo de dimensionamiento por omisión.



(c) Selección y edición de las compensaciones del área de trabajo.



(d) Selección y edición de las compensaciones de las herramientas.

Figura 4.5: Componentes editores de los datos de la categoría planificación de ruta

getDataCategoryID public String getDataCategoryID()
Propuesto por DataCategory, proporciona la cadena “pathPlanning” como identificador de la categoría.

getValue public String getValue(Id dataID) Propuesto por DataCategory, puede proporcionar el nombre clave del sistema métrico actual (los cuales son “international” o “english”), la compensación de trabajo (p.ej. “G54”) o el tipo de movimiento (los valores de la constante motionTypeOpt).

setCommandValue Las variantes son:

- public boolean setCommandValue(int id,int value)
- public boolean setCommandValue(int id,long value)

Propuesto por DataCategory, no se usa por que no hay comandos en esta categoria.

setDataValue Las variantes son:

- public abstract boolean setDataValue(int id,int value)
- public abstract boolean setDataValue(int id,long value)

Propuesto por DataCategory, establece los nuevos valores de los datos correspondientes al tipo de movimiento, plano cartesiano, sistema métrico por omisión, sistema métrico, modo de dimensionamiento, modo de dimensionamiento por omisión, valor de los ejes x, y, z de la posición destino, el índice de la compensación de trabajo activo, el índice de la compensación de la herramienta activa y el número total de compensaciones de trabajo.

applyChange public void applyChange()

Propuesto por DataCategory, confirma los cambios en los datos del sistema métrico por omisión, modo de dimensionamiento por omisión, la compensación de trabajo y la compensación de la herramienta, realizados en los componentes editables.

cancelChange public void cancelChange()

Propuesto por DataCategory, cancela los cambios en los datos del sistema métrico por omisión, modo de dimensionamiento por omisión, la compensación de trabajo y la compensación de la herramienta, realizados en los componentes editables.

initWorkOffset public void initWorkOffset(int x, int y, int z)

Inicializa los campos relacionados con la compensación de trabajo. Agrega un nuevo elemento a la lista `offsetAxes` y en ese orden es el comando de compensación de trabajo que le corresponde (p.ej. a la primera terna le corresponde el comando G54). Para administrar la información referente a las compensaciones de trabajo, se hace uso de las dos listas offsetAxes y offsetAxes_Aux. La lista offsetAxes almacena los datos en nanómetros, como los proporciona y maneja el código nativo. Por otra parte, la lista offsetAxes_Aux almacena la misma información pero en las unidades correspondientes al sistema métrico y tiene la finalidad de almacenar cambios sin afectar a los originales.

Parámetros:

`x` - Compensación de trabajo del eje x.

`y` - Compensación de trabajo del eje y.

`z` - Compensación de trabajo del eje z.

initToolOffset public void initToolOffset(int lenght, int diameter)

Inicializa los campos relacionados con las compensaciones de las herramientas. Agrega un nuevo elemento a la lista `offsetTools`, el orden en que son agregados debe corresponder con el orden del carrusel. Para administrar la información referente a las compensaciones de las herramienta, se hace uso de las dos listas `offsetTools` y `offsetTools_Aux`. La lista offsetTools almacena los datos en nanómetros, como los proporciona y maneja el código nativo. Por otra parte, la lista offsetTools_Aux almacena la misma información pero en las unidades correspondientes al sistema métrico y tiene la finalidad de almacenar cambios sin afectar a los originales.

Parámetros:

`lenght` - Compensación de la herramienta correspondiente a su longitud.

`diameter` - Compensación de la herramienta correspondiente a su diámetro.

getMetricSystem public static String getMetricSystem()
Proporciona el nombre clave del sistema métrico en uso. Este método tiene la finalidad de permitir la consulta del sistema métrico sin la necesidad de un ejemplar dentro de otras categorías.

getMotionType public static String getMotionType()
Proporciona el nombre clave del tipo de movimiento en uso. Este método tiene la finalidad de permitir la consulta del tipo de movimiento sin la necesidad de un ejemplar dentro de otras categorías.

IV.7.3 Métodos privados

checkNumberToolOffsets private void checkNumberToolOffsets()
Confirma que el número de elementos de compensación de trabajo en la lista concuerde con el índice máximo de las herramientas. Si el índice es mayor agrega datos de compensación inicializados igual a cero. Si el índice es menor remueve los datos sobrantes. Si son iguales no hace nada. Estas acciones no afectan a la base de datos a menos que se confirmen.

changeUnitsToNm private void changeUnitsToNm()
Realiza la conversión de unidades del sistema métrico en curso a nanómetros, de los datos que se encuentran en la lista `offsetAxes_Aux` y los almacena en la lista `offsetAxes`.

changeNmToCurrentSys private void changeNmToCurrentSys()
Realiza la conversión, nanómetros a las unidades del sistema métrico seleccionado, de los datos que se encuentran en la lista `offsetAxes` y los almacena en la lista `offsetAxes_Aux`.

toolAuxUnitsToNm private void toolAuxUnitsToNm()
Realiza la conversión, unidades del sistema métrico en curso a nanómetros, de los datos que se encuentran en la lista `offsetTools_Aux` y los almacena en la lista `offsetTools`.

toolNmToCurrentSys private void toolNmToCurrentSys()
Realiza la conversión, nanómetros a unidades del sistema métrico seleccionado, de los datos que se encuentran en la lista `offsetTools` y los almacena en la lista `offsetTools_Aux`.

getListener Las variantes son:

- private ItemListener getListener(final String[] aux, final JFormattedTextField workOffsetX, final JFormattedTextField workOffsetY, final JFormattedTextField workOffsetZ)

- private ItemListener getListener(final String[] aux, final JFormattedTextField toolOffsetL, final JFormattedTextField toolOffsetD)

Proporcionan oyentes de eventos de elementos para los seleccionadores múltiples usados para editar la compensaciones de trabajo y herramienta. El oyente vigila los eventos relacionados con los cambios de selección de cada elemento. Primero se determina el índice del elemento que origino el evento con el arreglo `aux`, después se determina si fue seleccionado o no. Si fue seleccionado se cargan en los campos editables los valores de la lista de compensaciones auxiliar y si por el contrario, se cambio de selección, el elemento correspondiente de la lista auxiliar toma el valor de los campos editables.

Parámetros:

`aux` - Arreglo de cadenas de texto que contiene los nombres que se visualizan en los seleccionadores múltiples correspondientes.

`workOffsetX` - Campo editable donde se muestra la compensación de trabajo del eje x.

`workOffsetY` - Campo editable donde se muestra la compensación de trabajo del eje y.

`workOffsetZ` - Campo editable donde se muestra la compensación de trabajo del eje z.

`toolOffsetL` - Campo editable donde se muestra la compensación de la longitud de la herramienta.

`toolOffsetD` - Campo editable donde se muestra la compensación del diámetro de la herramienta.

IV.7.4 Campos privados

listIndexWorkOffsetX private boolean[] listIndexWorkOffsetX
Arreglo que almacena el estado del campo editable del desplazamiento del área de trabajo correspondiente al eje X, true si a sido editado y false si no.

listIndexWorkOffsetY private boolean[] listIndexWorkOffsetY
Arreglo que almacena el estado del campo editable del desplazamiento del área de trabajo correspondiente al eje Y, true si a sido editado y false si no.

listIndexWorkOffsetZ private boolean[] listIndexWorkOffsetZ
Arreglo que almacena el estado del campo editable del desplazamiento del área de trabajo correspondiente al eje Z, true si a sido editado y false si no.

listIndexToolOffsetL private boolean[] listIndexToolOffsetL
Arreglo que almacena el estado del campo editable de la compensación de la longitud de la herramienta de corte, true si a sido editado y false si no.

listIndexToolOffsetD private boolean[] listIndexToolOffsetD
Arreglo que almacena el estado del campo editable de la compensación del radio de la herramienta de corte, true si a sido editado y false si no.

motionTypeOpt private static final String[] motionTypeOpt
Arreglo constante de cadenas que contiene los nombres clave de los tipos de movimiento. Las cadenas son “rapid”, “linear”, “clockwise” y “counterClockwise”.

cartesianPlaneOpt private static final String[] cartesianPlaneOpt
Arreglo constante de cadenas que contiene los nombres clave los planos cartesianos. Las cadenas son “xy”, “xz” y “yz”.

metricSystemOpt private static final String[] metricSystemOpt
Arreglo constante de cadenas que contiene los nombres clave de los sistemas métricos. Las cadenas son “international” y “english”.

dimModeOpt private static final String[] dimModeOpt
Arreglo constante de cadenas que contiene los nombres clave de los modos de dimensionamiento. las cadenas son “absolute” y “incremental”.

motionType private static Integer motionType
Almacena el tipo de movimiento seleccionado. El valor varía de 0 a 3 en el orden en que se definen en la constante `motionTypeOpt`.

cartesianPlane private static Integer cartesianPlane
Almacena el plano cartesiano seleccionado. El valor varía de 0 a 2 en el orden en que se definen en la constante `cartesianPlaneOpt`.

metricSystem private static Integer metricSystem
Almacena el sistema métrico seleccionado. El valor varía de 0 a 1 en el orden en que se definen en la constante `metricSystemOpt`.

international private JRadioButton international
Botón de radio de la opción del sistema métrico internacional.

english private JRadioButton english
Botón de radio de la opción del sistema métrico anglosajón.

dimMode private static Integer dimMode
Almacena el modo de dimensionamiento seleccionado. El valor varía de 0 a 1 en el orden en que se definen en la constante `dimModeOpt`.

absolute private JRadioButton absolute
Botón de radio de la opción del modo de dimensionamiento absoluto.

incremental private JRadioButton incremental
Botón de radio de la opción del modo de dimensionamiento incremental.

xATP private static Integer xATP
Almacena el valor de la posición destino correspondiente al eje x.

yATP private static Integer yATP
Almacena el valor de la posición destino correspondiente al eje y.

zATP private static Integer zATP
Almacena el valor de la posición destino correspondiente al eje z.

activeWOIndex private static Integer activeWOIndex
Almacena el valor del índice de compensación de trabajo activo.

activeTOIndex private static Integer activeTOIndex
Almacena el valor del índice de compensación de la herramienta activa.

defaultMetricSystem private static Integer defaultMetricSystem
Almacena el valor del sistema métrico por omisión. El valor varía de 0 a 1 en el orden en que se definen en la constante `metricSystemOpt`.

defaultDimMode private static Integer defaultDimMode
Almacena el modo de dimensionamiento por omisión. El valor varía de 0 a 1 en el orden en que se definen en la constante `dimModeOpt`.

numberWorkOffset private static Integer numberWorkOffset
Almacena el número total de compensaciones de trabajo.

offsetAxes private static ArrayList <Integer[]>offsetAxes
Lista de compensaciones de trabajo. Valores en nanómetros.

offsetTools private static ArrayList <Integer[]>offsetTools
Lista de compensaciones de herramienta. Valores en nanómetros.

offsetAxes_Aux private static ArrayList <Float[]>offsetAxes_Aux
Lista auxiliar de compensaciones de trabajo. Valores en milímetros o pulgadas.

offsetTools_Aux private static ArrayList <Float[]>offsetTools_Aux
Lista auxiliar de compensaciones de herramientas. Valores en milímetros o pulgadas.

motionTypeField private static JTextField motionTypeField
Campo donde se visualiza el tipo de movimiento.

cartesianPlaneField private static JTextField cartesianPlaneField
Campo donde se visualiza el plano cartesiano seleccionado.

metricSystemField private static JTextField metricSystemField
Campo donde se visualiza que sistema métrico se encuentra seleccionado.

dimModeField private static JTextField dimModeField
Campo donde se visualiza el modo de dimensionamiento.

xATPField private JFormattedTextField xATPField
Campo donde se visualiza la coordenada correspondiente al eje X de la posición destino.

yATPField private JFormattedTextField yATPField
Campo donde se visualiza la coordenada correspondiente al eje Y de la posición destino.

zATPField private JFormattedTextField zATPField
Campo donde se visualiza la coordenada correspondiente al eje Z de la posición destino.

xProgramZeroField private static JFormattedTextField xProgramZeroField
Campo donde se visualiza el valor correspondiente al eje X del cero pieza.

yProgramZeroField private static JFormattedTextField yProgramZeroField
Campo donde se visualiza el valor correspondiente al eje Y del cero pieza.

zProgramZeroField private static JFormattedTextField zProgramZeroField
Campo donde se visualiza el valor correspondiente al eje Z del cero pieza.

workOffsetX private JFormattedTextField workOffsetX
Campo donde se visualiza el valor correspondiente al eje X de la compensación de trabajo.

workOffsetY private JFormattedTextField workOffsetY
Campo donde se visualiza el valor correspondiente al eje Y de la compensación de trabajo.

workOffsetZ private JFormattedTextField workOffsetZ
Campo donde se visualiza el valor correspondiente al eje Z de la compensación de trabajo.

toolLenght private JFormattedTextField toolLenght
Campo donde se visualiza el valor de la compensación de la herramienta correspondiente a la longitud.

toolRadius private JFormattedTextField toolDiameter
Campo donde se visualiza el valor de la compensación de la herramienta correspondiente al diámetro.

IV.8. Clase AxisMotion

public class AxisMotion extends DataCategory

La clase AxisMotion agrupa los datos que describen el estado, movimiento y posición de los ejes. Extiende a la clase DataCategory.

IV.8.1 Constructor

AxisMotion public AxisMotion() Crea un objeto de tipo AxisMotion que representa una categoría de datos de la máquina. Se inicializan las etiquetas que contiene los iconos de estado de los interruptores límite.

IV.8.2 Métodos públicos

initData Las variantes son:

- public boolean initData(int id, int value)
- public boolean initData(int id, long value)

Propuesto por DataCategory, inicializa los campos del factor de manivela, velocidad de avance constante, modo de corte, avance de corte por omisión, avance de corte, corrección de avance de corte, avance de corte máximo, velocidad de avance constante máximo, corrección de avance constante, avance rápido de retorno a cero, velocidad de avance, corrección de la velocidad de avance, velocidad de avance máximo, rango de desplazamiento, punto de referencia de dirección de retorno, estado de los interruptores límite y posición actual.

getView public Component getView(Id viewID)
Propuesto por DataCategory, proporciona los componentes que muestran información respecto al modo de corte, la velocidad de avance, estado de los interruptores límite, posición actual, rango de desplazamiento y punto de referencia de dirección de retorno. Véase la tabla 4.4.

Tabla 4.4: Componentes que visualizan datos de la categoría movimiento de los ejes

Datos	Identificador	Figura
Modo de corte	Id.CUTTING_MODE	4.7a
Velocidad de avance	Id.FEEDRATE	4.7b
Interruptores límite	Id.AXIS_LIMIT_SWITCH	4.6
Posición actual	Id.AXIS_CURRENT_POSITION	4.7c
Rango de desplazamiento	Id.AXIS_MOTION_TRAVEL_RANGE	4.7d
Dirección de retorno	Id.AXIS_REFERENCE_POINT_RETURN_DIRECTION	4.7e

getEditor public Component getEditor(Id editorID)
Propuesto por DataCategory, proporciona componentes que permiten controlar y editar datos relacionados con la velocidad de avance de corte, constante, y del retorno a cero, rango de desplazamiento y la referencia de la dirección de retorno. Véase la tabla 4.5.

getDataCategoryID public String getDataCategoryID()
Propuesto por DataCategory, proporciona la cadena “axisMotion” como identificador de la categoría.

getValue public String getValue(Id dataID)
Propuesto por DataCategory, la categoría no lo usa.

setCommandValue Las variantes son:

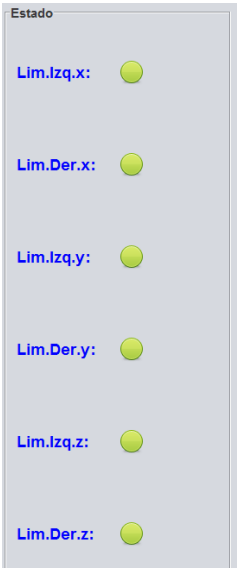


Figura 4.6: Componente que muestra el estado de los interruptores límite.

- public boolean setCommandValue(int id,int value)
- public boolean setCommandValue(int id,long value)

Propuesto por DataCategory, establece los nuevos valores de la velocidad de avance constante.

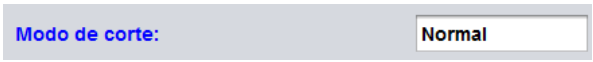
setDataValue Las variantes son:

- public abstract boolean setDataValue(int id,int value)
- public abstract boolean setDataValue(int id,long value)

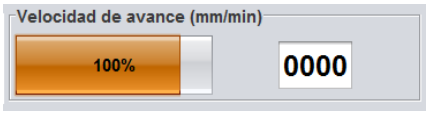
Propuesto por DataCategory, establece los nuevos valores de los datos correspondientes al modo de corte, velocidad de avance de corte, corrección de la velocidad de avance rápida, constante y de corte, interruptores límite y posición actual.

applyChange public void applyChange()
Propuesto por DataCategory, confirma los cambios en los datos de la velocidad máxima y por omisión del avance de corte, la velocidad de avance constante y su límite máximo, velocidad de avance rápida, velocidad máxima y rápida del avance de retorno a cero, rango de desplazamiento y la dirección de retorno, realizados en los componentes editables.

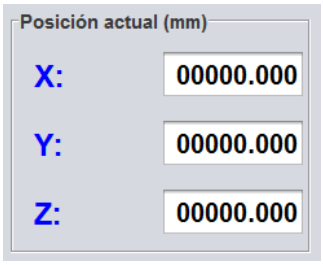
cancelChange public void cancelChange()
Propuesto por DataCategory, cancela los cambios en los datos de la velocidad máxima y por omisión del avance de corte, la velocidad de avance constante y su límite máximo, velocidad de avance rápida, velocidad máxima y rápida del avance de retorno a cero, rango de desplazamiento y la dirección de retorno, realizados en los componentes editables.



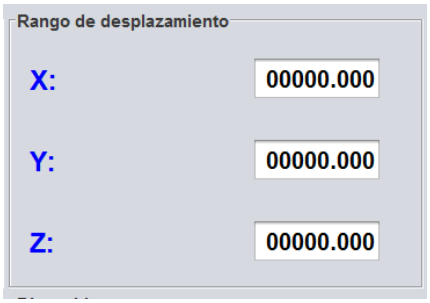
(a) Modo de corte.



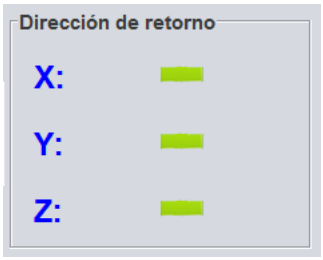
(b) Velocidad de avance y su corrección.



(c) Posición actual.



(d) Rango de desplazamiento.



(e) Punto de referencia de la dirección de retorno.

Figura 4.7: Componentes visualizadores de la categoría movimiento de los ejes

IV.8.3 Métodos privados

getFeedrate private int getFeedrate()

Proporciona la velocidad de avance correspondiente al tipo de movimiento que este activo o al modo de operación.

getOverride private int getOverride()

Proporciona la corrección de la velocidad de avance correspondiente al tipo de movimiento que este activo o al modo de operación.

IV.8.4 Enumeraciones

HandwheelF private enum HandwheelF

Enumeración de los factores de avance de la manivela. Los valores son `HF_1`, `HF_10` y `HF_100`.

HandwheelA private enum HandwheelA

Enumeración con tres elementos que representan los tres ejes, los cuales la manivela puede controlar. Los valores son `HA_X`, `HA_Y` y `HA_Z`.

IV.8.5 Campos privados

cuttingModeOpt private static final String[] cuttingModeOpt

Arreglo constante de cadenas que contiene definidos los nombres clave de los distintos

Tabla 4.5: Componentes editables y controles de la categoría movimiento de los ejes

Datos	Identificador	Figura
Retorno a cero	Id.ZERO_RETURN	4.8a
Corrección de la velocidad del avance de corte	Id.CUTTING_FEEDRATE_OVERRIDE	4.8b
Corrección de la velocidad de avance rápido	Id.RAPID_TRAVERSE_RATE_OVERRIDE	4.8c
Velocidad de avance constante	Id.JOG_FEEDRATE	4.8d
Corrección de la velocidad de avance constante	Id.JOG_FEEDRATE_OVERRIDE	4.8e
Factor de avance de la manivela	Id.HANDWHEEL_FACTOR	4.8f
Eje de la manivela	Id.HANDWHEEL_AXIS	4.8g
Velocidad por omisión y máxima del avance de corte	Id.CUTTING_DATA_EDITOR	4.9a
Velocidad máxima y por omisión del avance constante	Id.JOG_DATA_EDITOR	4.9b
Velocidad de avance rápido y su valor máximo	Id.RTR_DATA_EDITOR	4.9c
Velocidad de avance del retorno a cero	Id.ZRTR_DATA_EDITOR	4.9d
Rango de desplazamiento	Id.AXIS_MOTION_TRAVEL_RANGE	4.9e
Referencia de dirección de retorno	Id.AXIS_REFERENCE_POINT_RETURN_DIRECTION	4.9f
Avance constante	Id.JOG_AXES	4.8h

modos de operación. Los valores son “normal”, “exactStop”, “automaticCornerOverride” y “tapping”.

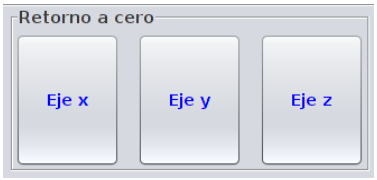
axisHLSOpt private static final String[] axisHLSOpt
Arreglo constante de cadenas que contiene los nombres clave de los posibles estados de los interruptores límite. Los valores son “enabled” y “disabled”

arprdOpt private String[] arprdOpt
Arreglo de cadenas que contiene definidos nombres clave correspondientes al punto de referencia de la dirección de retorno. Los valores son “negative” y “positive”.

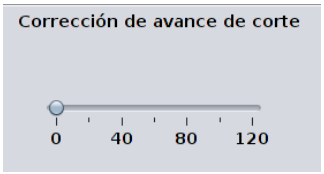
handwheelF private static HandwheelF handwheelF
Ejemplar de la enumeración HandwheelF.

handwheelA private static HandwheelA handwheelA
Ejemplar de la enumeración HandwheelA.

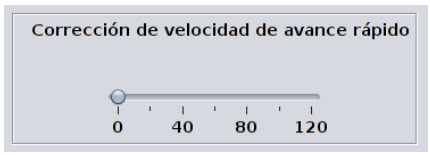
cuttingMode private static Integer cuttingMode
Campo que almacena el valor correspondiente al modo de corte en uso.



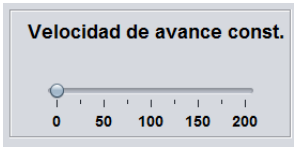
(a) Control del retorno a cero.



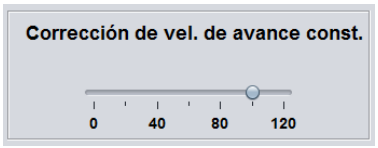
(b) Control de la corrección de la velocidad de corte.



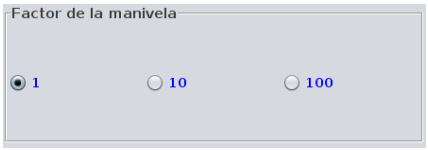
(c) Control de la corrección de la velocidad de avance rápido.



(d) Control de la velocidad de avance constante.



(e) Control de la corrección de avance constante.



(f) Control del factor de avance de la manivela.

(g) Control de selección del eje de la manivela.

(h) Control de los ejes en el modo de movimiento constante.

Figura 4.8: Controles de la categoría movimiento de los ejes

maxCutFeedrate private static Integer maxCutFeedrate
Campo que almacena el valor de la velocidad máxima de avance de corte.

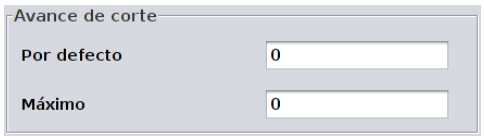
maxCutField private JFormattedTextField maxCutField
Campo de texto editable correspondiente a la velocidad máxima de avance de corte.

cutFeedrate private static Integer cutFeedrate
Campo que almacena el valor de la velocidad de avance de corte.

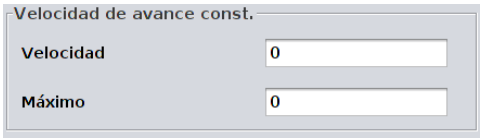
dfltCutFeedrate private static Integer dfltCutFeedrate
Campo que almacena el valor de la velocidad de avance de corte por omisión.

dfltCutField private JFormattedTextField dfltCutField
Campo de texto editable correspondiente a la velocidad de avance de corte por omisión.

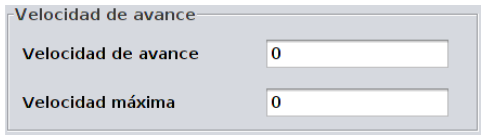
cutFeedrateO private static Integer cutFeedrateO
Campo que almacena el valor de la corrección de la velocidad de avance de corte.



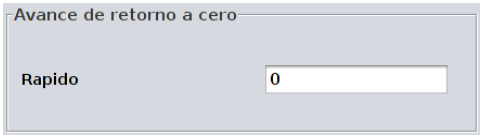
(a) Editor de la velocidad por omisión y máxima del avance de corte.



(b) Editor de la velocidad por omisión y máxima del avance constante.

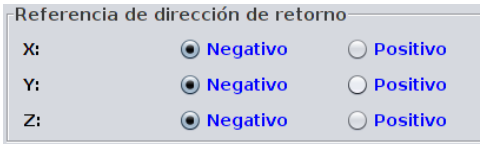


(c) Editor de la velocidad de avance y su valor máximo.



(d) Editor de la velocidad de avance del retorno a cero.

(e) Editor del rango de desplazamiento.



(f) Editor del punto de referencia de dirección de retorno.

Figura 4.9: Componentes editables de la categoría movimiento de los ejes

maxRapidTR private static Integer maxRapidTR
Campo que almacena el valor máximo de la velocidad de avance rápido.

maxRapidTRField private JFormattedTextField maxRapidTRField
Campo de texto editable correspondiente a la velocidad máxima del avance rápido.

rapidTR private static Integer rapidTR
Campo que almacena el valor de la velocidad de avance rápido.

rapidTRField private JFormattedTextField rapidTRField
Campo de texto editable correspondiente a la velocidad del avance rápido.

rapidTRO private static Integer rapidTRO
Campo que almacena el valor de corrección de la velocidad de avance rápido.

jogFeedrate private static Integer jogFeedrate
Campo que almacena el valor de la velocidad del avance constante.

dfltJogField private JFormattedTextField dfltJogField
Campo de texto editable correspondiente a la velocidad de avance constante por omisión.

maxJogFeedrate private static Integer maxJogFeedrate
Campo que almacena el valor máximo de la velocidad del avance constante.

maxJogField private JFormattedTextField maxJogField
Campo de texto editable correspondiente a la velocidad máxima de avance constante.

jogFeedrateO private static Integer jogFeedrateO
Campo que almacena el valor de corrección de la velocidad del avance constante.

xALHLS private static Integer xALHLS
Estado del interruptor límite izquierdo del eje X correspondiente al hardware. Los valores posibles son 0 o 1, activado o desactivado respectivamente.

xARHLS private static Integer xARHLS
Estado del interruptor límite derecho del eje X correspondiente al hardware. Los valores posibles son 0 o 1, activado o desactivado respectivamente.

yALHLS private static Integer yALHLS
Estado del interruptor límite izquierdo del eje Y correspondiente al hardware. Los valores posibles son 0 o 1, activado o desactivado respectivamente.

yARHLS private static Integer yARHLS
Estado del interruptor límite derecho del eje Y correspondiente al hardware. Los valores posibles son 0 o 1, activado o desactivado respectivamente.

zALHLS private static Integer zALHLS
Estado del interruptor límite izquierdo del eje Z correspondiente al hardware. Los valores posibles son 0 o 1, activado o desactivado respectivamente.

zARHLS private static Integer zARHLS
Estado del interruptor límite derecho del eje Z correspondiente al hardware. Los valores posibles son 0 o 1, activado o desactivado respectivamente.

xAMTR private static Integer xAMTR
Campo que almacena el rango de desplazamiento correspondiente al eje X.

yAMTR private static Integer yAMTR
Campo que almacena el rango de desplazamiento correspondiente al eje Y.

zAMTR private static Integer zAMTR
Campo que almacena el rango de desplazamiento correspondiente al eje Z.

xACP private static Integer xACP
Campo que almacena el valor de la posición actual correspondiente al eje X.

yACP private static Integer yACP
Campo que almacena el valor de la posición actual correspondiente al eje Y.

zACP private static Integer zACP
Campo que almacena el valor de la posición actual correspondiente al eje Z.

fastZRTR private static Integer fastZRTR
Campo que almacena el valor de velocidad de avance rápido del retorno a cero.

fastZRTRField private JFormattedTextField fastZRTRField
Campo de texto editable correspondiente a la velocidad de avance rápido del retorno a cero.

slowZRTR private static Integer slowZRTR
Campo de texto editable correspondiente a la velocidad de avance lento del retorno a cero.

slowZRTRField private JFormattedTextField slowZRTRField
Campo de texto editable correspondiente a la velocidad de avance lento del retorno a cero.

xARPRD private static Integer xARPRD
Campo que almacena el valor del punto de referencia de la dirección de retorno correspondiente al eje X. Los valores son 0 y 1, correspondientes a si es "negative" o "positive".

yARPRD private static Integer yARPRD
Campo que almacena el valor del punto de referencia de la dirección de retorno correspondiente al eje Y. Los valores son 0 y 1, correspondientes a si es "negative" o "positive".

zARPRD private static Integer zARPRD
Campo que almacena el valor del punto de referencia de la dirección de retorno correspondiente al eje Z. Los valores son 0 y 1, correspondientes a si es "negative" o "positive".

xARPRD_Aux private static Integer xARPRD_Aux
Campo auxiliar para la edición del valor del campo xARPRD.

yARPRD_Aux private static Integer yARPRD_Aux
Campo auxiliar para la edición del valor del campo yARPRD.

zARPRD_Aux private static Integer zARPRD_Aux
Campo auxiliar para la edición del valor del campo zARPRD.

xNButton private JRadioButton xNButton
Botón de radio correspondiente a la dirección de retorno negativa del eje X.

xPButton private JRadioButton xPButton
Botón de radio correspondiente a la dirección de retorno positiva del eje X.

yNButton private JRadioButton yNButton
Botón de radio correspondiente a la dirección de retorno negativa del eje Y.

yPButton private JRadioButton yPButton
Botón de radio correspondiente a la dirección de retorno positiva del eje Y.

zNButton private JRadioButton zNButton
Botón de radio correspondiente a la dirección de retorno negativa del eje Z.

zPButton private JRadioButton zPButton
Botón de radio correspondiente a la dirección de retorno positiva del eje Z.

cuttingModeField private JTextField cuttingModeField
Campo de texto no editable que visualiza el nombre del modo de corte en uso.

fField private static JFormattedTextField fField
Campo de texto no editable con formato numérico que visualiza la velocidad de avance.

fBar private static JProgressBar fBar
Barra que visualiza la corrección de la velocidad de avance.

aLHLSIcon private static JLabel[] aLHLSIcon
Arreglo de etiquetas destinadas a almacenar los iconos indicadores del estado de los interruptores limite correspondientes al lado izquierdo.

aRHLSIcon private static JLabel[] aRHLSIcon
Arreglo de etiquetas destinadas a almacenar los iconos indicadores del estado de los interruptores limite correspondientes al lado derecho.

xACPField private JFormattedTextField xACPField
Campo de texto no editable con formato numérico que visualiza el valor de la posición actual correspondiente al eje X.

yACPField private JFormattedTextField yACPField
Campo de texto no editable con formato numérico que visualiza el valor de la posición actual correspondiente al eje Y.

zACPField private JFormattedTextField zACPField
Campo de texto no editable con formato numérico que visualiza el valor de la posición actual correspondiente al eje Z.

xAMTRField private static JFormattedTextField xAMTRField
Campo de texto no editable con formato numérico que visualiza el rango de desplazamiento correspondiente al eje X.

yAMTRField private static JFormattedTextField yAMTRField
Campo de texto no editable con formato numérico que visualiza el rango de desplazamiento correspondiente al eje Y.

zAMTRField private static JFormattedTextField zAMTRField
Campo de texto no editable con formato numérico que visualiza el rango de desplazamiento correspondiente al eje Z.

IV.9. Clase Spindle

public class Spindle extends DataCategory

La clase Spindle agrupa los datos que controlan el movimiento y velocidad del husillo. Extiende a la clase DataCategory.

IV.9.1 Constructor

Spindle public Spindle() Crea un objeto de tipo Spindle que representa una categoría de datos de la máquina.

IV.9.2 Métodos públicos

initData Las variantes son:

- public boolean initData(int id, int value)
- public boolean initData(int id, long value)

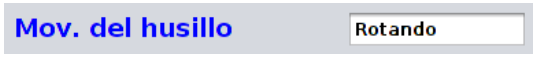
Inicializa los campos de la dirección de rotación del husillo, estado del movimiento del husillo, velocidad del husillo, corrección de la velocidad del husillo, velocidad mínima y máxima del husillo y corrección mínima y máxima del husillo.

getView public Component getView(Id viewID)
Propuesto por DataCategory, proporciona los componentes que muestran información respecto a la velocidad y su corrección del husillo, así como el movimiento y dirección del husillo. Véase la tabla 4.6.

Tabla 4.6: Componentes que visualizan datos de la categoría husillo

Datos	Identificador	Figura
Corrección, velocidad y dirección del husillo	Id.SPINDLE	4.10a
Movimiento del husillo	Id.SPINDLE_MOTION	4.10b
Dirección del husillo	Id.SPINDLE_DIRECTION	4.10c

(a) Velocidad, corrección y dirección del husillo.



(b) Estado del movimento del husillo.

(c) Dirección del movimiento del husillo.

Figura 4.10: Componentes visualizadores de la categoría husillo

getEditor public Component getEditor(Id editorID)
Propuesto por DataCategory, proporciona componentes que permiten editar y controlar la dirección de rotación y movimiento del husillo, la corrección y la velocidad del husillo, así como sus límites mínimo y máximo. Véase la tabla 4.7.

getDataCategoryID public String getDataCategoryID()
Propuesto por DataCategory, proporciona la cadena “spindle” como identificador de la categoría.

Tabla 4.7: Componentes editables de la categoría husillo

Datos	Identificador	Figura
Corrección de la velocidad del husillo	Id.SPINDLE_SPEED_OVERRIDE	4.11a
Dirección del husillo	Id.SPINDLE_DIRECTION	4.11b
Movimiento del husillo	Id.SPINDLE_MOTION	4.11c
Velocidad del husillo	Id.SPINDLE_SPEED	4.11d
Valores mínimo y máximo para la velocidad y su corrección	Id.SPINDLE	4.11e

getValue public String getValue(Id dataID) Propuesto por DataCategory, no es usado por la categoría Spindle.

setCommandValue Las variantes son:

- public boolean setCommandValue(int id,int value)
- public boolean setCommandValue(int id,long value)

Propuesto por DataCategory, establece los nuevos valores de los datos correspondientes al movimiento y dirección del husillo, además de la velocidad y corrección del husillo.

setDataValue Las variantes son:

- public abstract boolean setDataValue(int id,int value)
- public abstract boolean setDataValue(int id,long value)

Propuesto por DataCategory, este método no es usado por la categoría.

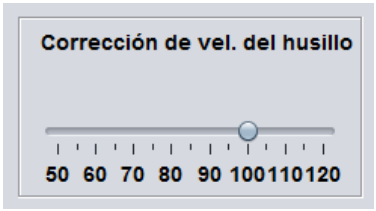
applyChange public void applyChange()
Propuesto por DataCategory, confirma los cambios en los datos de la velocidad mínima y máxima del husillo, así como la corrección mínima y máxima de la velocidad del husillo, realizados en los componentes editables.

cancelChange public void cancelChange()
Propuesto por DataCategory, cancela los cambios en los datos de la velocidad mínima y máxima del husillo, así como la corrección mínima y máxima de la velocidad del husillo, realizados en los componentes editables.

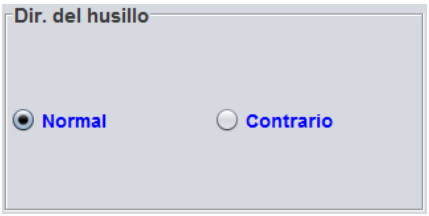
IV.9.3 Campos privados

spindleMtnOpt private static final String[] spindleMtnOpt
Arreglo constante de cadenas de texto que almacena los nombres clave de las opciones del movimiento del husillo. Los valores son “rotate” y “stop”.

spindleDirOpt private static final String[] spindleDirOpt
Arreglo constante de cadenas de texto que almacena los nombres clave de las opciones de la dirección del husillo. Los valores son “normal” y “reverse”.

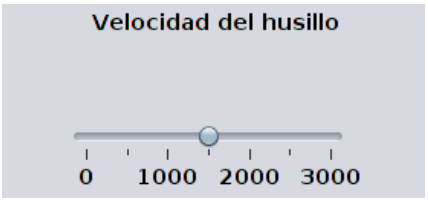


(a) Control de la corrección de la velocidad del husillo.

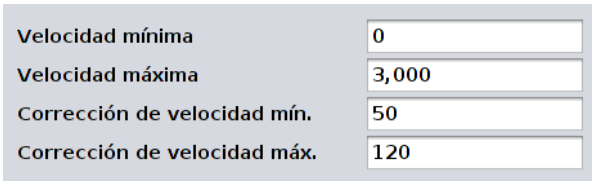


(b) Control de la dirección del husillo.

(c) Control que permite encender o apagar el husillo.



(d) Control de la velocidad del husillo.



(e) Editor de los valores mínimo y máximo de la velocidad y corrección del husillo.

Figura 4.11: Componentes editables y controles de la categoría husillo

spindleDir private static Integer spindleDir
Campo que almacena el valor correspondiente a la dirección seleccionada de rotación del husillo. Los valores son 0 y 1, en dirección de las manecillas del reloj y en contra, respectivamente.

spindleMtn private static Integer spindleMtn
Campo que almacena el valor correspondiente al movimiento del husillo. Los valores son 0 y 1, rotar y detener, respectivamente.

spindleSpdO private static Integer spindleSpdO
Campo que almacena el valor de la corrección de la velocidad del husillo.

minSpindleSpdO private static Integer minSpindleSpdO
Campo que almacena el valor mínimo de la corrección de la velocidad del husillo.

maxSpindleSpdO private static Integer maxSpindleSpdO
Campo que almacena el valor máximo de la corrección de la velocidad del husillo.

spindleSpd private static Integer spindleSpd
Campo que almacena el valor de la velocidad del husillo.

minSpindleSpd private static Integer minSpindleSpd
Campo que almacena el valor mínimo de la velocidad del husillo.

maxSpindleSpd private static Integer maxSpindleSpd
Campo que almacena el valor máximo de la velocidad del husillo.

minSSField private JFormattedTextField minSSField
Campo de texto editable correspondiente al valor mínimo de la velocidad del husillo.

maxSSField private JFormattedTextField maxSSField
Campo de texto editable correspondiente al valor máximo de la velocidad del husillo.

maxSSOField private JFormattedTextField maxSSOField
Campo de texto editable correspondiente al valor mínimo de la corrección de la velocidad del husillo.

minSSOField private JFormattedTextField minSSOField
Campo de texto editable correspondiente al valor máximo de la corrección de la velocidad del husillo.

sField private static JFormattedTextField sField
Campo de texto no editable que visualiza la velocidad del husillo.

sBar private static JProgressBar sBar
Barra que visualiza el valor de la corrección de la velocidad del husillo.

spindleDirIcon private static JLabel spindleDirIcon
Etiqueta que contiene el icono que representa la dirección en función.

spindleMtnField private static JTextField spindleMtnField
Campo de texto no editable que visualiza el estado del movimiento del husillo.

spindleDirField private static JTextField spindleDirField
Campo de texto no editable que visualiza el nombre de la dirección del husillo en función.

IV.10. Clase Atc

public class Atc extends DataCategory

La clase Atc agrupa los datos relacionados con el administrador de herramientas automático. Extiende a la clase DataCategory.

IV.10.1 Constructor

Atc public Atc() Crea un objeto de tipo Atc que representa una categoría de datos de la máquina.

IV.10.2 Métodos públicos

initData Las variantes son:

- public boolean initData(int id, int value)
- public boolean initData(int id, long value)

Inicializa los campos de la capacidad de herramientas del carrusel, índice máximo de las herramientas, tipo de selección de la herramienta, índice de la herramienta seleccionada y la siguiente.

getView public Component getView(Id viewID)

Propuesto por DataCategory, proporciona los componentes que muestran información respecto a la capacidad del carrusel, índice máximo, tipo de selección, índice de la herramienta actual y la siguiente. Véase la tabla 4.8.

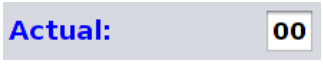
Tabla 4.8: Componentes visualizadores de la categoría administrador de herramientas automático

Datos	Identificador	Figura
Capacidad de herramientas del carrusel	Id.TOOL_MAGAZINE_CAPACITY	4.12a
Índice máximo de las herramienta	Id.MAXIMUM_TOOL_INDEX	4.12b
Tipo de selección de las herramientas	Id.TOOL_SELECTION_TYPE	4.12c
Índice de la herramienta seleccionada	Id.SELECTED_TOOL_INDEX	4.12d
Índice de la herramienta siguiente	Id.NEXT_TOOL_INDEX	4.12e

(a) Capacidad del carrusel de herramientas.

(b) Índice máximo de las herramientas.

(c) Modo de selección de las herramientas.



(d) Índice de la herramienta seleccionada.

(e) Índice de la siguiente herrramienta.

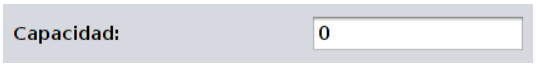
Figura 4.12: Componentes visualizadores de la categoría administrador de herramientas automático

getEditor public Component getEditor(Id editorID)

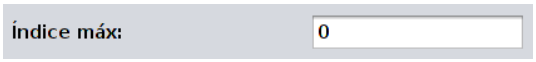
Propuesto por DataCategory, proporciona componentes que permiten editar la capacidad total del carrusel, el índice máximo de las herramientas presentes, elegir el tipo de selección y el índice de la herramienta seleccionada. Véase la tabla 4.9.

Tabla 4.9: Componentes editables de la categoría administrador de herramientas automático

Datos	Identificador	Figura
Capacidad del carrusel	Id.TOOL_MAGAZINE_CAPACITY	4.13a
Índice máximo de las herramientas	Id.MAXIMUM_TOOL_INDEX	4.13b
Tipo de selección	Id.TOOL_SELECTION_TYPE	4.13c
Índice de la herramienta actual	Id.SELECTED_TOOL_INDEX	4.13d

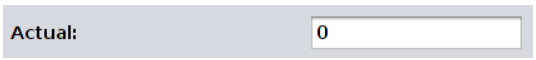


(a) Editor la capacidad del carrusel.



(b) Editor del índice máximo de las herramientas.

(c) Editor del tipo de selección de las herramientas.



(d) Editor del índice de la herramienta seleccionada.

Figura 4.13: Componentes editables del administrador de herramientas automático

getDataCategoryID public String getDataCategoryID()
Propuesto por DataCategory, proporciona la cadena “atc” como identificador de la categoría.

getValue public String getValue(Id dataID) Propuesto por DataCategory, proporciona el índice máximo de las herramientas.

setCommandValue Las variantes son:

- public boolean setCommandValue(int id,int value)
- public boolean setCommandValue(int id,long value)

Propuesto por DataCategory, este método no es usado por la categoría.

setDataValue Las variantes son:

- public abstract boolean setDataValue(int id,int value)
- public abstract boolean setDataValue(int id,long value)

Propuesto por DataCategory, establce nuevos valores correspondientes a los datos de la capacidad de herramientas del carrusel, índice máximo de las herramientas, tipo de selección de la herramienta, índice de la herramienta seleccionada y la siguiente.

applyChange public void applyChange()
Propuesto por DataCategory, confirma los cambios en los datos de la capacidad del carrusel, índice máximo, tipo de selección de la herramienta e índice de la herramienta seleccionada, realizados en los componentes editables.

cancelChange public void cancelChange()
Propuesto por DataCategory, cancela los cambios en los datos de la capacidad del carrusel, índice máximo, tipo de selección de la herramienta e índice de la herramienta seleccionada, realizados en los componentes editables.

getMaxToolIndex public static int getMaxToolIndex()
Proporciona el índice máximo de las herramientas.

IV.10.3 Campos privados

toolSTypeOpt private static final String[] toolSTypeOpt
Arreglo de cadenas de texto constante que almacena los nombres clave de los tipo de selección de las herramientas. Los valores son "fixed" y "random".

toolMCapacity private static Integer toolMCapacity
Campo que almacena el valor correspondiente a la capacidad máxima del carrusel de herramientas.

toolMCapacityField private JFormattedTextField toolMCapacityField
Campo de texto no editable que visualiza la capacidad del carrusel.

toolSType private static Integer toolSType
Campo que almacena el valor correspondiente al tipo de selección de la herramienta. Los valores son 0 y 1, fijo y aleatorio, respectivamente.

toolSType_Aux private static Integer toolSType_Aux
Campo auxiliar para la edición del campo toolSType.

fixedButton private JRadioButton fixedButton
Botón de radio correspondiente a la opción de selección fija de la herramienta.

randomButton private JRadioButton randomButton
Botón de radio correspondiente a la opción de selección aleatoria de la herramienta.

selectedToolIndex private static Integer selectedToolIndex
Campo que almacena el valor correspondiente al índice de la herramienta seleccionada.

selectedToolIndexField private JFormattedTextField selectedToolIndexField
Campo de texto no editable que visualiza el índice de la herramienta seleccionada.

nextToolIndex private static Integer nextToolIndex
Campo que almacena el valor correspondiente al índice de la herramienta siguiente.

nextToolIndexField private JFormattedTextField nextToolIndexField
Campo de texto no editable que visualiza el índice de la herramienta siguiente.

maxToolIndex private static Integer maxToolIndex
Campo que almacena el valor correspondiente al índice máximo de las herramientas en el carrusel.

maxToolIndexField private JFormattedTextField maxToolIndexField
Campo de texto no editable que visualiza el índice máximo de las herramientas en el carrusel.

IV.11. Clase ToolHolder

public class ToolHolder extends DataCategory

La clase ToolHolder agrupa los datos referentes a las dimensiones de las herramientas en forma general. Extiende a la clase DataCategory.

IV.11.1 Constructor

ToolHolder public ToolHolder() Crea un objeto de tipo ToolHolder que representa una categoría de datos de la máquina.

IV.11.2 Métodos públicos

initData Las variantes son:

- public boolean initData(int id, int value)
- public boolean initData(int id, long value)

Inicializa los campos de la longitud y diámetro máximos de las herramientas.

getView public Component getView(Id viewID)
Propuesto por DataCategory, la categoría no cuenta con componentes que solo visualizan información.

getEditor public Component getEditor(Id editorID)
Propuesto por DataCategory, proporciona componentes que permiten editar la longitud y diámetro máximo de las herramientas. Véase la tabla 4.10.

Tabla 4.10: Componentes editables de la categoría soporte de la herramienta

Datos	Identificador	Figura
Longitud y diámetro máximos de las herramientas	Id.TOOL_HOLDER_DATA_EDITOR	4.14

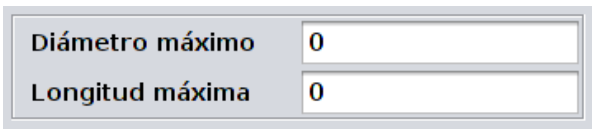


Figura 4.14: Componente que permite editar las dimensiones de las herramientas.

getDataCategoryID public String getDataCategoryID()
Propuesto por DataCategory, proporciona la cadena “toolHolder” como identificador de la categoría.

getValue public String getValue(Id dataID) Propuesto por DataCategory, este método no es usado por la categoría.

setCommandValue Las variantes son:

- public boolean setCommandValue(int id,int value)
- public boolean setCommandValue(int id,long value)

Propuesto por DataCategory, este método no es usado por la categoría.

setDataValue Las variantes son:

- public abstract boolean setDataValue(int id,int value)
- public abstract boolean setDataValue(int id,long value)

Propuesto por DataCategory, establece nuevos valores correspondientes a la longitud y diámetro máximos de las herramientas.

applyChange public void applyChange()
Propuesto por DataCategory, confirma los cambios en los datos de la longitud y diámetro máximos de las herramientas, realizados en los componentes editables.

cancelChange public void cancelChange()
Propuesto por DataCategory, cancela los cambios en los datos de la longitud y diámetro máximos de las herramientas, realizados en los componentes editables.

IV.11.3 Campos privados

maxToolDiameter private static Integer maxToolDiameter
Campo que almacena el valor correspondiente al diámetro máximo de las herramientas.

diameterField private JFormattedTextField diameterField
Campo de texto editable correspondiente al diámetro máximo de las herramientas.

maxToolLength private static Integer maxToolLength
Campo que almacena el valor correspondiente a la longitud máxima de las herramientas.

lengthField private JFormattedTextField lengthField
Campo de texto editable correspondiente a la longitud máxima de las herramientas.

IV.12. Clase Devices

public class Devices extends DataCategory

La clase Devices agrupa los datos relacionados al número y estado de dispositivos presentes en el sistema. Extiende a la clase DataCategory.

IV.12.1 Constructor

Devices public Devices() Crea un objeto de tipo Devices que representa una categoría de datos de la máquina.

IV.12.2 Métodos públicos

initData Las variantes son:

- public boolean initData(int id, int value)
- public boolean initData(int id, long value)

Inicializa los campos del número total de dispositivos y su estado de cada uno.

getView public Component getView(Id viewID)
Propuesto por DataCategory, proporciona los componentes que muestran información respecto al estado de los dispositivos. Véase la tabla 4.11.

Tabla 4.11: Componentes que visualizan datos de la categoría dispositivos

Datos	Identificador	Figura
Estado de los dispositivos	Id.DEVICES	4.15

Figura 4.15: Componente que muestra el estado de los dispositivos.

getEditor public Component getEditor(Id editorID)
Propuesto por DataCategory, proporciona componentes que permiten establecer el estado de los dispositivos. Véase la tabla 4.12.

getDataCategoryID public String getDataCategoryID()
Propuesto por DataCategory, proporciona la cadena “devices” como identificador de la categoría.

Tabla 4.12: Componentes editables de la categoría dispositivos

Datos	Identificador	Figura
Estado de los dispositivos	Id.DEVICES	4.16

Figura 4.16: Componente que permite elegir el estado de los dispositivos.

getValue public String getValue(Id dataID) Propuesto por DataCategory, este método no es usado por la categoría.

setCommandValue Las variantes son:

- public boolean setCommandValue(int id,int value)
- public boolean setCommandValue(int id,long value)

Propuesto por DataCategory, establece nuevos valores correspondientes al estado de los dispositivos y cambia el icono de estado.

setDataValue Las variantes son:

- public abstract boolean setDataValue(int id,int value)
- public abstract boolean setDataValue(int id,long value)

Propuesto por DataCategory, este método no es usado por la categoría.

applyChange public void applyChange()
Propuesto por DataCategory, este método no es usado por la categoría.

cancelChange public void cancelChange()
Propuesto por DataCategory, este método no es usado por la categoría.

IV.12.3 Campos privados

deviceOpt private static final String[] deviceOpt
Arreglo de cadenas de texto constante que almacena los nombres clave de los distintos estados en que se pueden encontrar los dispositivos. Los valores son “off”,“on” y “auto”.

numberDevices private static Integer numberDevices
Campo que almacena el valor correspondiente al número de dispositivos.

lightStatus private static Integer lightStatus
Campo que almacena el valor correspondiente al estado del la luz.

coolerStatus private static Integer coolerStatus
Campo que almacena el valor correspondiente al estado del refrigerante

lightIcon private static JLabel lightIcon
Etiqueta que contiene al icono que representa el estado de la luz.

coolerIcon private static JLabel coolerIcon
Etiqueta que contiene al icono que representa el estado del refrigerante.

IV.13. Clase ProgramExecution

public class ProgramExecution extends DataCategory

La clase ProgramExecution agrupa los datos que determinan la forma en que se ejecuta un programa. Extiende a la clase DataCategory.

IV.13.1 Constructor

ProgramExecution public ProgramExecution() Crea un objeto de tipo ProgramExecution que representa una categoría de datos de la máquina.

IV.13.2 Métodos públicos

initData Las variantes son:

- public boolean initData(int id, int value)
- public boolean initData(int id, long value)

Inicializa los campos del estado del ciclo de maquinado, ejecución del bloque, paro opcional, omisión del bloque, ejecución de prueba, bloqueo principal, bloqueo del eje Z y bloqueo de los ejes XY.

getView public Component getView(Id viewID)
Propuesto por DataCategory, proporciona el componente que muestra el nombre del programa. En el programa no se usa este componente, el nombre del programa se muestra en el marco de una lámina.

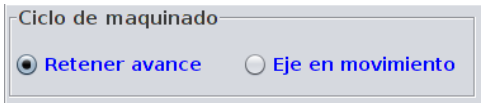
getEditor public Component getEditor(Id editorID)
Propuesto por DataCategory, proporciona componentes que permiten elegir el estado del ciclo de maquinado, ejecución del bloque, el paro opcional, la omisión del bloque, ejecución de prueba, bloqueo principal, bloqueo del eje Z y el bloqueo de los ejes XY. Véase la tabla 4.13.

getDataCategoryID public String getDataCategoryID()
Propuesto por DataCategory, proporciona la cadena "programExecution" como identificador de la categoría.

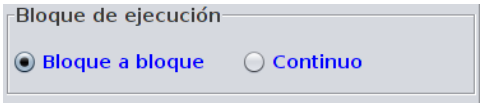
getValue public String getValue(Id dataID)
Propuesto por DataCategory, proporciona el nombre del programa que se encuentra cargado.

Tabla 4.13: Componentes editables de la categoría ejecución del programa

Datos	Identificador	Figura
Estado del ciclo de maquinado	Id.MACHINING_CYCLE_STATUS	4.17a
Ejecución del bloque	Id.BLOCK_EXECUTION	4.17b
Paro opcional	Id.OPTIONAL_STOP	4.17c
Omitir bloque	Id.BLOCK_SKIP	4.17d
Corrida de prueba	Id.DRY_RUN	4.17e
Bloqueo principal	Id.MST_LOCK	4.17f
Bloqueo de los ejes xy	Id.XY_AXES_LOCK	4.17g
Bloqueo del eje z	Id.Z_AXIS_NEGLECT	4.17h



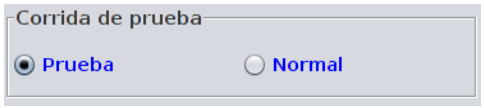
(a) Control del estado del ciclo de maquinado.



(b) Control del modo de ejecución del bloque.

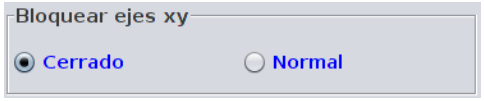
(c) Control del paro opcional.

(d) Control de omisión del bloque.



(e) Control de corrida de prueba.

(f) Control del bloqueo principal.



(g) Bloqueo de los ejes XY.

(h) Bloqueo del eje Z.

Figura 4.17: Controles de la categoría ejecución del programa

setCommandValue Las variantes son:

- public boolean setCommandValue(int id,int value)
- public boolean setCommandValue(int id,long value)

Propuesto por DataCategory, este método no es usado por la categoría.

setDataValue Las variantes son:

- public abstract boolean setDataValue(int id,int value)
- public abstract boolean setDataValue(int id,long value)

Propuesto por DataCategory, establece nuevos valores correspondientes a los datos del ciclo de maquinado, ejecución del bloque, paro opcional, omisión del bloque, ejecución de prueba, bloqueo principal, bloqueo del eje Z y bloqueo de los ejes XY.

applyChange public void applyChange()
Propuesto por DataCategory, este método no es usado por la categoría.

cancelChange public void cancelChange()
Propuesto por DataCategory, este método no es usado por la categoría.

IV.13.3 Campos privados

machCycleStatusOpt private static final String[] machCycleStatusOpt
Arreglo constate de cadenas que contiene los nombres clave de las opciones del ciclo de maquinado. Los valores son “feedhold” y “axesMotion”.

blockExecOpt private static final String[] blockExecOpt
Arreglo constate de cadenas que contiene los nombres clave de las opciones de ejecución de bloque. Los valores son “singleBlock” y “continuous”.

optStopOpt private static final String[] optStopOpt
Arreglo constate de cadenas que contiene los nombres clave de las opciones del paro opcional. Los valores son “stop” y “skip”.

blockSkipOpt private static final String[] blockSkipOpt
Arreglo constate de cadenas que contiene los nombres clave de las opciones de la omisión del bloque. Los valores son “skip” y “read”.

dryRunOpt private static final String[] dryRunOpt
Arreglo constate de cadenas que contiene los nombres clave de las opciones de la corrida de prueba. Los valores son “dry” y “normal”.

mstLockOpt private static final String[] mstLockOpt
Arreglo constate de cadenas que contiene los nombres clave de las opciones del bloqueo principal. Los valores son “locked” y “normal”.

xyAxesLockOpt private static final String[] xyAxesLockOpt
Arreglo constate de cadenas que contiene los nombres clave de las opciones del bloqueo de los ejes XY. Los valores son “locked” y “normal”.

zAxisNeglectOpt private static final String[] zAxisNeglectOpt
Arreglo constate de cadenas que contiene los nombres clave de las opciones del bloqueo del eje Z. Los valores son “locked” y “normal”.

jobName private static String jobName
Campo que almacena el nombre del programa cargado.

machCycleStatus private static Integer machCycleStatus
Campo que almacena el valor del estado del ciclo de maquinado. El valor varía entre 0 y 1, retener avance y eje en movimiento respectivamente, como en la constante `machCycleStatusOpt`.

blockExec private static Integer blockExec
Campo que almacena el valor correspondiente al estado de ejecución del bloque. El valor varía entre 0 y 1, bloque a bloque y continuo respectivamente, como en la constante `blockExecOpt`.

optStop private static Integer optStop
Campo que almacena el valor del estado del paro opcional. El valor varía entre 0 y 1, detener y omitir respectivamente, como en la constante `optStopOpt`.

blockSkip private static Integer blockSkip
Campo que almacena el valor del estado de la omsión del bloque. El valor varía entre 0 y 1, omitir y leer respectivamente, como en la constante `blockSkipOpt`.

dryRun private static Integer dryRun
Campo que almacena el valor del estado de la corride de prueba. El valor varía entre 0 y 1, prueba y normal respectivamente, como en la constante `dryRunOpt`.

mstLock private static Integer mstLock
Campo que almacena el valor del estado del bloqueo principal. El valor varía entre 0 y 1, bloquear y normal respectivamente, como en la constante `mstLockOpt`.

xyAxesLock private static Integer xyAxesLock
Campo que almacena el valor del estado del bloqueo de los ejes XY. El valor varía entre 0 y 1, bloquear y normal respectivamente, como en la constante `xyAxesLockOpt`.

zAxisNeglect private static Integer zAxisNeglect
Campo que almacena el valor del estado del bloqueo del eje Z. El valor varía entre 0 y 1, bloquear y normal respectivamente, como en la constante `zAxisNeglectOpt`.

jobNameField private static JTextField jobNameField
Campo de texto no editable donde se visualiza el nombre del programa que se encuentra cargado.

IV.14. Clase Data

public class Data extends DataCategory

La clase Data agrupa todas la categorías de datos y proporciona funciones para el acceso. Extiende a la clase DataCategory.

IV.14.1 Constructor

Data public Data()
Crea un nuevo objeto de tipo Data inicializando todas las categorías.

IV.14.2 Métodos públicos

initData Las variantes son:

- public boolean initData(int id, int value)
- public boolean initData(int id, long value)

Propuesto por DataCategory, comunica los valores iniciales a la categoría de datos correspondiente.

getView public Component getView(Id viewID)
Propuesto por DataCategory, proporciona el componente especificado en el parámetro.

getEditor public Component getEditor(Id editorID) Propuesto por DataCategory, proporciona el componente especificado en el parámetro.

getDataCategoryID public String getDataCategoryID()
Propuesto por DataCategory, proporciona la cadena "data" como identificador.

getValue public String getValue(Id dataID)
Propuesto por DataCategory, proporciona el valor correspondiente al dato especificado en el parámetro, en forma de cadena de texto.

setCommandValue Las variantes son:

- public boolean setCommandValue(int id,int value)
- public boolean setCommandValue(int id,long value)

Propuesto por DataCategory, comunica los cambios realizados por comandos a la categoría de datos correspondiente.

setDataValue Las variantes son:

- public abstract boolean setDataValue(int id,int value)
- public abstract boolean setDataValue(int id,long value)

Propuesto por DataCategory, comunica cambios realizados en datos a la categoría correspondiente.

applyChange public void applyChange()
Propuesto por DataCategory, confirma los cambios realizados a través de los campos editables.

cancelChange public void cancelChange()
Propuesto por DataCategory, cancela los cambios realizados a través de los campos editables.

getCategory Las variantes son:

- public DataCategory getCategory(Category value)

- public DataCategory getCategory(int value)

Proporciona un objeto de la categoría de dato correspondiente al valor establecido en el parámetro.

Parámetros:

`value` - Valor correspondiente a la categoría que se busca.

getTotalCategory public int getTotalCategory()
Proporciona el número total de categorías existentes.

initWorkOffset public static void initWorkOffset(int x,int y,int z)
Comunica los valores iniciales de las compensaciones de trabajo.

Parámetro:

`x` - compensación del eje X.

`y` - compensación del eje Y.

`z` - compensación del eje Z.

initToolOffset public static void initToolOffset(int lenght, int diameter)
Comunica los valores iniciales de las compensación de las herramientas presentes.

Parámetro:

`lenght` - compensación de la longitud de la herramienta.

`diameter` - compensación del diámetro de la herramienta.

IV.14.3 Campos privados

machineOperation private MachineOperation machineOperation
Ejemplar de la categoría de datos “operación de la máquina”.

pathPlanning private PathPlanning pathPlanning
Ejemplar de la categoría de datos “planificación de la ruta”.

axisMotion private AxisMotion axisMotion
Ejemplar de la categoría de datos “movimiento de los ejes”.

programExecution private ProgramExecution programExecution
Ejemplar de la categoría de datos “ejecución del programa”.

spindle private Spindle spindle
Ejemplar de la categoría de datos “husillo”.

atc private Atc atc
Ejemplar de la categoría de datos “cambio automático de herramienta”.

devices private Devices devices
Ejemplar de la categoría de datos “dispositivos”.

toolHolder private ToolHolder toolHolder
Ejemplar de la categoría de datos “soporte de la herramienta”.

IV.15. Clase Mode

public class Mode

La clase Mode es la base para definir un modo de operación proporcionando métodos de utilidad para armar las pantallas y algunos componentes especiales. Cada clase derivada de Mode cuenta con un botón de radio para informar y establecer el modo de operación. Además, se deben definir los componentes que serán agregados en las láminas superior e inferior al ser seleccionado el modo.

La lámina superior se divide en las zonas centro, norte, sur, este y oeste como se muestra en figura 4.18 [11]. Este comportamiento es por que la lámina cuenta con un encargado de disposición de borde (BorderLayout). El lugar norte esta destinada al nombre del modo de operación y el estado del paro de emergencia. Las zonas restantes están disponibles para los otros componentes. En estas zonas los componentes se agregan primero a una lámina rotulada con borde.

La lámina inferior también se divide en cinco zonas como se muestra en la figura 4.19. Pero en esta lámina, cuando los componentes se agregan en la zona centro, son agrupados en solapas según la categoría a la que pertenecen. Los componentes especiales sin categoría pueden colocarse en cualquier otra zona.

Figura 4.18: Lugares en que se divide el panel superior. Las zonas son: norte, sur, este, oeste y centro.

IV.15.1 Constructor

Mode Las variantes son:

[11] Véase la enumeración constante Place

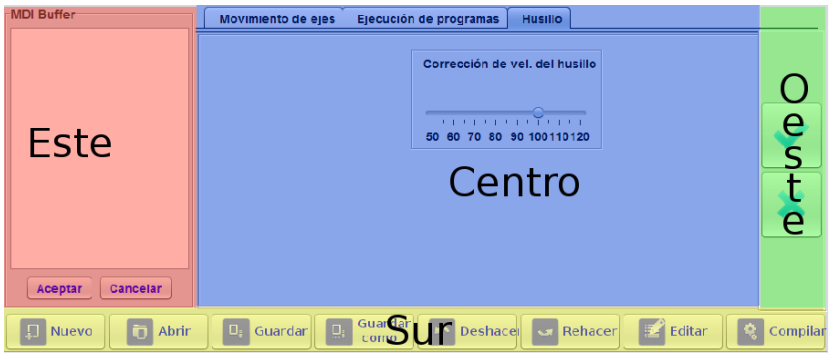


Figura 4.19: Lugares en que se divide el panel inferior. Las zonas son: norte, sur, este, oeste y centro.

- public Mode(String name, final Integer mode_)
- public Mode()

Crea un objeto de tipo Model, el cual representa un modo de operación.

Parámetros:

`name` - nombre clave del modo de operación.

`mode_` - valor del modo de operación.

IV.15.2 Métodos públicos

initRadioButton public void initRadioButton(String name, boolean isSelected, ActionListener listener)
Inicializa el botón de radio cuyo fin es cargar el modo de operación correspondiente al ser seleccionado.

Parámetros:

`name` - nombre clave del modo de operación.

`isSelected` - establece si el bóton inicia seleccionado.

`listener` - establece el oyente de eventos. En este se establecen todas las tareas necesarias para desplegar el modo de operación en pantalla.

addTopPanels public void addTopPanels(String namePanel, Place place, GridBagConstraints position, Data.Id[] description, AnotherComponent[] anotherComponents)
Agrega la descripción de una nueva lámina con marco, la cual se agrega a la lámina superior. Por ejemplo:

```
addTopPanels(
            "cut",
            Mode.Place.SOUTH,
            new GridBagConstraints(
                0,
```

```
            0,
            1,
            1,
            1,
            0,
            GridBagConstraints.CENTER,
            GridBagConstraints.BOTH,
            new Insets(0, 0, 0, 0),
            0,
            0
        ),
        new Data.Id[] {
            Data.Id.MOTION_TYPE,
            Data.Id.CARTESIAN_PLANE,
            Data.Id.DIMENSIONING_MODE,
            Data.Id.CUTTING_MODE,
            Data.Id.SPINDLE_DIRECTION,
            Data.Id.SPINDLE_MOTION
        },
        null
);
```

Este código crea la lámina de la figura 4.20, donde el nombre clave “cut” corrresponde a la cadena de texto “Corte” y el lugar donde se coloca es en la zona sur, en la posición (0,0)[12]. Se agregan seis componentes de distintas categorías y no es necesario especificar ningún componente especial, por lo que se anula el ultimo parámetro.

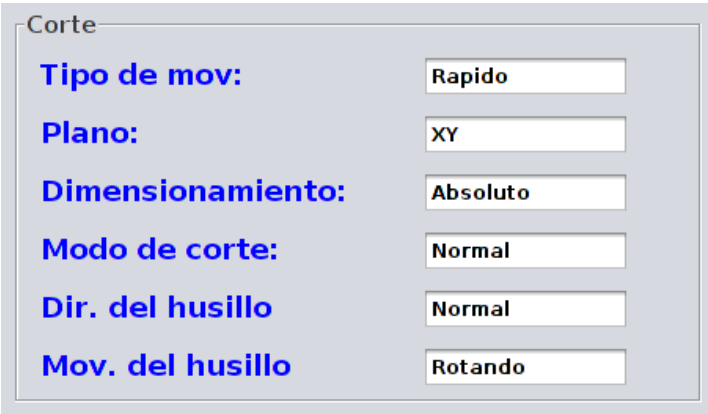


Figura 4.20: Ejemplo de una lámina para la pantalla superior.

Parámetros:

`namePanel` - nombre del rótulo del marco de la lámina.

[12] Véase la clase GridBagLayout y GridBagConstraints

`place` - lugar en la cual se colocara.[13]

`position` - dentro de la zona se especifica una posición especifica.

`description` - es un arreglo de identificadores de componentes. El orden en que aparecen en el arreglo, es el orden en que son agregados a la lámina.[14]

`anotherComponents` - un arreglo donde se establece un componente sin categoría. Este arreglo se debe especificar si en el parámetro `description` se agrego un `Data.OTHER`. Si es el caso, solo este componente se toma en cuenta para ser agregado a la lámina, todos los demás son descartados.

addBottomPanels public void addBottomPanels(String namePanel, Place place, Data.Id[] description, AnotherComponent[] anotherComponents,GridLayout layout)
Agrega la descripción de una nueva solapa u otro tipo de componente especial, la cual se agrega a la lámina inferior editable. Por ejemplo:

```
public void setDescriptionEditablePanel()
{
    addBottomPanels(
        "axisMotion",
        Mode.Place.CENTER,
        new Data.Id[] {
            Data.Id.OTHER,
            Data.Id.ZERO_RETURN,
            Data.Id.OTHER,
            Data.Id.OTHER,
            Data.Id.OTHER,
            Data.Id.OTHER
        },
        new AnotherComponent[] {
            AnotherComponent.PANEL,
            AnotherComponent.PANEL,
            AnotherComponent.PANEL,
            AnotherComponent.PANEL,
            AnotherComponent.PANEL
        },
        new GridLayout(2,3)
    );
}
```

Este código crea la solapa de la figura 4.21, donde el nombre clave “axisMotion” corresponde a la cadena de texto “Movimiento de ejes” y el lugar donde se coloca es en

[13] Figura 4.18
[14] Véase DataCategory y sus extensiones

la zona centro. Se agregan seis componentes, donde solo uno pertenece a la categoría y el resto son láminas. Por ultimo, se establece una malla de 2x3 para colocar los componentes.

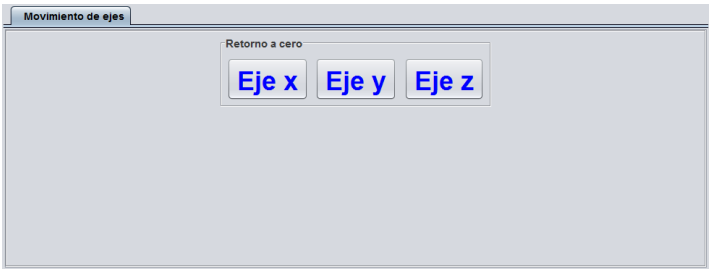


Figura 4.21: Ejemplo de una solapa para la pantalla inferior.

Parámetros:

`namePanel` - nombre del rótulo de la solapa. Solo se toma en cuenta si se agrega una solapa en la zona centro de la lámina.

`place` - lugar en el cual se colocara.

`description` - es un arreglo de elementos de tipo Data. El orden en que aparecen en el arreglo, es el orden en que son agregados a la lámina de la solapa.

`anotherComponents` - arreglo donde se establecen los componentes sin categoría. Este arreglo se debe especificar si en el parametro `description` se agrego un `Data.OTHER`. En otro caso debe ser declarado nulo.

`layout` - establece un administrador de disposición GridLayout lo cual permite un acomodo más equitativo del espacio o también permite rellenar el espacio sobrante con láminas extra. Estas láminas extra se deben indicar por separado en el arreglo `description` como `Data.OTHER` y determinar en el arreglo `anotherComponents` como `AnotherComponent.PANEL`. Se puede declarar como nulo si se quiere usar el administrador de disposición por omisión FlowLayout.

getViewPanel public static JPanel getViewPanel()
Proporciona la lámina superior correspondiente a los componentes no editables que solo visualizan información.

getEditablePanel public static JPanel getEditablePanel()
Proporciona la lámina inferior correspondiente a los componentes editables.

getButton public AbstractButton getButton()
Proporciona el botón de radio que establece el modo de operación.

getFileName public static String getFileName()
Proporciona al código nativo el nombre del archivo necesario ara realizar tareas como “abrir archivo”, “guardar”, “guardar como”, etc.

getNewFileName public static String getNewFileName()

Proporciona al código nativo el nuevo nombre del archivo para realizar tareas como "renombrar archivo" o "guardar como".

getLineText public static String getLineText()

Proporciona linea a linea el texto del editor. Cada "linea" en realidad son veinte caracteres, donde no necesariamente se termina en el carácter especial de nueva linea. Puede generar una excepción si el rango de texto solicitado excede el área valida del documento.

setLineText public static void setLineText(String line)

Establece una nueva linea de texto al final del área de texto activa. Puede generar una excepción si la posición donde se busca insertar el texto no es valida.

Parámetros:

`line` - cadena de texto a ser agregada.

setMode public void setMode()

Establece los componentes respectivos del modo de operación en las láminas superior e inferior. También selecciona el botón de radio si no a sido seleccionado y establece el comando respectivo en las categorías.

updateTable public static void updateTable()

Actualiza la tabla del administrador de archivos.

setDescriptionViewPanel abstract public void setDescriptionViewPanel()

Se propone este método para establecer la descripción de la lámina superior del modo de operación correspondiente. Se declaran todas las descripciones de los componentes con la función `addTopPanels`. Este método es llamado dentro de `setMode`.

setDescriptionEditablePanel abstract public void setDescriptionEditablePanel()

Se propone este método para establecer la descripción de la lámina inferior del modo de operación correspondiente. Se declaran todas las las descripciones de los componentes con la función `addBottomPanels`. Este método es llamado dentro de `setMode`.

IV.15.3 Métodos privados

pressButton private void pressButton()

Si el botón de radio no se encuentra seleccionado, lo selecciona.

setViewComponents private void setViewComponents()

Establece los componentes de la lámina superior. Esencialmente, lo que se hace es borrar el contenido de todas las láminas, después se entra a un ciclo donde se crea una lamina con marco y rótulo, se obtienen todos los componentes de la descripción y se colocan en la lámina anterior. Por ultimo la lámina se agrega a la zona y posición establecidas. Todo esto se repite hasta que ya no se cuenta con más elementos descriptivos, terminando con la actualización del panel superior apara visualizar los cambios.

Lo anterior puede variar dependiendo del componente, pues existen casos donde la lámina tendrá, además del titulo rotulado, información sobre las unidades de magnitud o información de las compensaciones del área de trabajo. También, algunos componentes especiales ya contaran con su propia lámina con marco y rótulo, en estos casos se deshecha la lámina anterior y solo se agrega esta ultima.

setEditableComponents private void setEditableComponents()

Establece los componentes de la lámina inferior. Esencialmente, lo que hace es borrar el contenido de la lámina y crear una lámina contenedora de solapas, después se entra en un ciclo donde se crea una lámina, a la cual se le agregan todos los componentes de la descripción. Por ultimo la lámina se agrega a la zona establecida, si la posición elegida es en el centro, se agrega como solapa. Todo esto se repite hasta que ya no se cuenta con más elementos descriptivos, terminando con la actualización del panel inferior apara visualizar los cambios.

getAnotherComponent private Component getAnotherComponent(
AnotherComponent another)

Proporciona componentes que realizan tareas especificas, los cuales no entran en una categoría de datos. Son componentes encargados de la edición de archivos, exploración de directorios, captura de comandos ingresados manualmente, visualizar código de tareas realizadas por la máquina, validación de cambios y láminas auxiliares. En la tabla 4.14 se muestran los componentes, su identificador y la referencia de la figura correspondiente.

Parámetros:

`another` - identificador del componente[15].

Tabla 4.14: Componentes especiales.

Componente	Identificador	Figura
Lámina auxiliar	PANEL	
Búfer IMD	MDI_BUFFER	4.22
Área donde se visualizan las tareas del programa	PROGRAM_EXECUTION_ AREA	4.23
Área de texto del editor	EDITOR_AREA	4.24
Botones de tareas del editor	TASKBAR	4.25
Tabla del administrador de archivos	FILE_MANAGER	
Botones de tareas del administrador	TASKBAR_FILE_MANAGER	4.26
Botones para validar cambios de campos editables	VALIDATE_CHANGE	4.27

[15] Véase la enumeración AnotherComponent

Figura 4.22: Componente donde se pueden ingresar datos para el control de la máquina.

initEditor private void initEditor()
: Inicializa el área de texto del editor, se le agrega un oyente de cambios de edición que permite hacer y deshacer modificaciones en el texto. También se le agrega un oyente de foco, el cual, sirve para recuperar el foco mientras se escribe con el teclado virtual.

editorPanel private void editorPanel()
: Borra el contenido, agrega el área de texto y actualiza la lámina del editor. Este método es de utilidad cuando se busca abrir un archivo[16].

browserPanel private void browserPanel()
: Borra el contenido, agrega el explorador de archivos y actualiza la lámina del editor. Este método es de utilidad cuando se busca abrir un archivo[17].

taskEditorPanel private void taskEditorPanel()
: Borra el contenido, agrega los botones de tareas de edición y actualiza la lámina de tareas del editor. Este método es de utilidad cuando se busca abrir o guardar con un nombre distinto un archivo[18].

taskBrowserPanel private void taskBrowserPanel()
: Borra el contenido, agrega los botones de tareas de administración y actualiza la lámina de tareas del explorador de directorios. Este método es de utilidad cuando se busca renombrar o borrar un archivo[19].

[16] Véase el método browserPanel
[17] Véase el método editorPanel
[18] Véase los métodos confirmDialog y nameDialog
[19] Véase los métodos confirmDialog y nameDialog

Figura 4.23: Componente donde se muestran las líneas de código que se ejecutan.

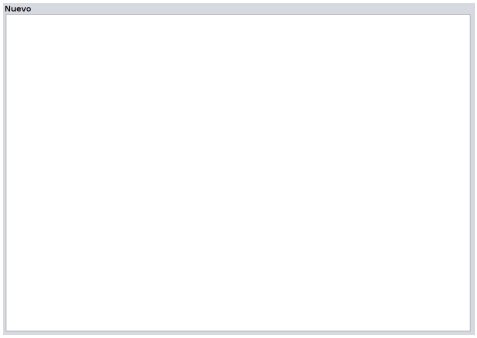


Figura 4.24: Componente donde se visualizan y editan los archivos de programas de la máquina.

confirmDialog private void confirmDialog(final String option,final ActionListener okListener,final ActionListener cancelListener)
Agrega, en la lámina de tareas del explorador de archivos, un dialogo de confirmación para tareas como la de borrar archivos.

Parámetros:

`option` - rótulo alternativo para el botón de confirmación de la tarea.

`okListener` - oyente de eventos que contiene las acciones a realizar cuando se confirma la tarea.

`cancelListener` - oyente de eventos que contiene las acciones a realizar cuando se cancela la tarea.

nameDialog private void nameDialog(final ActionListener okListener, final ActionListener cancelListener)
Agrega, en la lámina de tareas del explorador de archivos, un dialogo donde se solicita un nombre para realizar tareas como la de renombrar archivos.

Figura 4.25: Componente con botones para realizar tareas de edición.

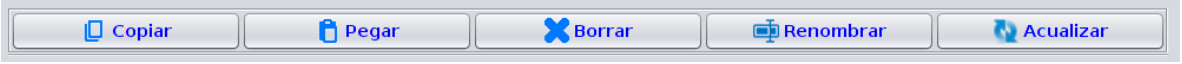


Figura 4.26: Componente con botones para realizar tareas de administración de archivos.

Parámetros:

`okListener` - oyente de eventos que contiene las acciones a realizar cuando se confirma la tarea.

`cancelListener` - oyente de eventos que contiene las acciones a realizar cuando se cancela la tarea.

undoAction private void undoAction()
Deshace cambios en el área de texto del editor.

redoAction private void redoAction()
Rehace cambios en el área de texto del editor.

IV.15.4 Enumeraciones

Place public static enum Place
Enumeración de los lugares en que se dividen las láminas superior e inferior. Los valores son `NORTH`, `SOUTH`, `EAST`, `WEST` y `CENTER`.

AnotherComponent public static enum AnotherComponent
Enumeración de los identificadores referentes a los componentes especiales. Véase la tabla 4.14.

Task private static enum Task
Enumeración de los identificadores de las tareas del explorador y administrador de archivos. Los valores son `COPY 0x00`, `OPEN 0x01`, `SAVE 0x02`, `SAVEAS 0x03`, `BROWSER_REFRESH 0x04`, `DELETE 0x05` y `RENAME 0x06`.

IV.15.5 Campos privados

indexLine private static int indexLine
Campo que almacena el número de caracteres que se ha enviado al código nativo. Es usado para tener un control al guardar archivos, registra el número de caracteres enviados y se compara con el número total para saber cuantos faltan.

undo private static UndoManager undo
Ejemplar de la clase UndoManager encargado de administrar los cambios de un documento.

Figura 4.27: Componente con dos botones que confirman o cancelan cambios realizados en campos editables.

nameEditFile private static JLabel nameEditFile
Etiqueta del nombre del archivo que esta siendo editado.

isSaved private static boolean isSaved
Campo que confirma si el archivo esta guardado.

editorArea private static JTextArea editorArea
Área de texto del editor.

nameField private JTextField nameField
Campo de texto para las tareas de edición y administración de archivos, como por ejemplo "guardar como", aquí se almacena el nuevo nombre.

browserTable private static JTable browserTable
Tabla del explorador de directorios donde se listan los archivos.

editorPanel private static JPanel editorPanel
Lámina del editor de texto.

panelTaskBrowser private static JPanel panelTaskBrowser
Lámina donde se colocan las tareas del explorador de archivos.

scrollTable private static JScrollPane scrollTable
Lámina con barras de desplazamiento para colocar la tabla del explorador de archivos.

panelTable private static JPanel panelTable
Lámina de la tabla del explorador de archivos.

fileName private static String fileName
Cadena de texto que almacena el nombre del archivo editado.

newFileName private static String newFileName
Cadena de texto auxiliar para tareas de edición y administración de archivos.

panelNorth private static JPanel panelNorth
Lámina que agrupa los componentes destinados a la zona norte de la lámina superior.

panelSouth private static JPanel panelSouth
Lámina que agrupa los componentes destinados a la zona sur de la lámina superior.

panelWest private static JPanel panelWest
Lámina que agrupa los componentes destinados a la zona oeste de la lámina superior.

panelEast private static JPanel panelEast
Lámina que agrupa los componentes destinados a la zona este de la lámina superior.

panelCenter private static JPanel panelCenter
Lámina que agrupa los componentes destinados a la zona centro de la lámina superior.

viewPanel private static JPanel viewPanel
Lámina superior destinada a los componentes que solo muestran información y no son editables, con alguna excepción.

editablePanel private static JPanel editablePanel
Lámina inferior destinada a los componentes editables y controles.

radioB private JRadioButton radioB
Botón de radio para establecer el modo de operación

dscrptnTop private ArrayList<Dscrptn>dscrptnTop
Lista descriptiva de la pantalla superior del modo de operación.

dscrptnBottom private ArrayList<Dscrptn>dscrptnBottom
Lista descriptiva de la pantalla inferior del modo de operación.

mode private Integer mode
Campo que almacena el valor numérico correspondiente al modo de operación.

data private static Data data
Ejemplar de la clase Data para acceder a las categorías de datos.

programArea private static JTextArea programArea
Área de texto de visualización del programa del modo memoria.

IV.15.6 Clases internas

Dscrptn private class Dscrptn
La clase interna Dscrptn tiene la finalidad de almacenar toda la información concerniente a las especificaciones de las pantallas de los modos de operación.

Constructor

Dscrptn public Dscrptn()
Crea un nuevo objeto de tipo Dscrptn e inicializa todos los campos de la clase.

Métodos públicos

setNamePanel public void setNamePanel(String namePanel_)
Establece el nombre para una lámina.

setPlace public void setPlace(Place place_)
Establece un lugar de los cinco definidos en la enumeración `Place`.

setPositionPanel public void setPositionPanel(GridBagConstraints positionPanel_)
Establece una posición dentro de la zona determinada en el método setPlace.

setDescriptionPanel public void setDescriptionPanel(Data.Id[] descriptionPanel_)
Establece el arreglo de identificadores de los componentes que se buscan colocar en la lámina.

getNamePanel public String getNamePanel()
Proporciona el nombre para una lámina.

getPlace public Place getPlace()
Proporciona el lugar donde se quiere colocar la lámina.

getStringPlace public String getStringPlace()
Proporciona una cadena de texto correspondiente al lugar elegido. A cada elemento de la enumeración Place le corresponde una constante de la clase BorderLayout, las cuales son cadenas usadas al momento de agregar elementos a contenedores con este manejador de disposición.

getPositionPanel public GridBagConstraints getPositionPanel()
Proporciona la posición de la lámina dentro del lugar elegido.

getDescriptionPanel public Data.Id[] getDescriptionPanel()
Proporciona el arreglo de identificadores de los componentes que se busca agregar a la lámina.

setAnotherDescription public void setAnotherDescription(AnotherComponent another_)
Establece el arreglo de identificadores de los componentes especiales.

getAnotherDescription public AnotherComponent getAnotherDescription()
Proporciona, uno a uno, los identificadores de los componentes especiales.

setLayout public void setLayout(LayoutManager layout_)
Establece un manejador de disposición para la lámina.

getLayout public LayoutManager getLayout()
Proporciona el manejador de disposición para la lámina.

Campos

namePanel private String namePanel
Campo que almacena el nombre para la lámina.

positionPanel private GridBagConstraints positionPanel
Campo que almacena la posición de la lamina.

descriptionPanel private Data.Id[] descriptionPanel
Arreglo de identificadores que almacena los componentes que se desean colocar.

place private Place place
Arreglo que almacena el lugar donde sera puesta la lámina.

another private ArrayList<AnotherComponent>another
Arreglo de identificadores de otros componentes especiales.

layout private LayoutManager layout

Campo que almacena el manejador de disposición para la lámina si no se quiere el manejador por omisión.

IV.16. Clase ZeroReturnMode

public class ZeroReturnMode extends Mode

La clase ZeroReturnMode corresponde al modo de operación de retorno a cero. Extiende a la clase Mode.

IV.16.1 Constructor

ZeroReturnMode public ZeroReturnMode()

Crea un objeto de tipo ZeroReturnMode que representa al modo de operación retorno a cero. Se establece el nombre clave y valor del comando del modo de operación.

IV.16.2 Métodos públicos

setDescriptionViewPanel public void setDescriptionViewPanel()

Propuesto por Mode, establece los componentes cuyos identificadores son:

Id.AXIS_CURRENT_POSITION	Id.MAXIMUM_TOOL_INDEX
Id.PROGRAM_ZERO	Id.TOOL_SELECTION_TYPE
Id.SPINDLE	Id.SELECTED_TOOL_INDEX
Id.FEEDRATE	Id.NEXT_TOOL_INDEX
Id.AXIS_LIMIT_SWITCH	Id.OPERATION_MODE
Id.TOOL_MAGAZINE_CAPACITY	Id.EMERGENCY_STOP

La lámina superior resultante se muestra en la figura 4.28.

Figura 4.28: Captura de la ventana superior del modo de retorno a cero.

setDescriptionEditablePanel public void setDescriptionEditablePanel()

Propuesto por Mode, establece un solo componente, correspondiente al identificador Id.ZERO_RETURN junto con cinco láminas auxiliares para rellenar el espacio sobrante. En la figura 4.29 muestra la lámina resultante.

Figura 4.29: Captura de la lámina inferior del modo de retorno a cero.

IV.17. Clase JogMode

public class JogMode extends Mode

La clase JogMode corresponde al modo de operación de avance constante. Extiende a la clase Mode.

IV.17.1 Constructor

JogMode public JogMode()

Crea un objeto de tipo JogMode que representa al modo de avance constante. Se establece el nombre clave y valor del comando del modo de operación.

IV.17.2 Métodos públicos

setDescriptionViewPanel public void setDescriptionViewPanel()

Propuesto por Mode, establece los componentes cuyos identificadores son:

Id.EMERGENCY_STOP
Id.OPERATION_MODE
Id.ATP
Id.AXIS_LIMIT_SWITCH
Id.AXIS_CURRENT_POSITION
Id.AXIS_MOTION_TRAVEL_RANGE
Id.SPINDLE
Id.DEVICES
Id.FEEDRATE

La lámina superior resultante se muestra en la figura 4.30.

setDescriptionEditablePanel public void setDescriptionEditablePanel()

Propuesto por Mode, establece los componentes cuyos identificadores son:

Figura 4.30: Captura de la lámina superior del modo de avance constante.

Id.JOG_FEEDRATE
Id.JOG_FEEDRATE_OVERRIDE
Id.JOG_AXES
Id.SPINDLE_DIRECTION
Id.SPINDLE_MOTION
Id.SPINDLE_SPEED_OVERRIDE
Id.DEVICES

La lámina inferior resultante, y sus solapas, se muestran en la figura 4.31.

IV.18. Clase MdiMode

public class MdiMode extends Mode

La clase MdiMode corresponde al modo de operación de avance constante. Extiende a la clase Mode.

IV.18.1 Constructor

MdiMode public MdiMode()

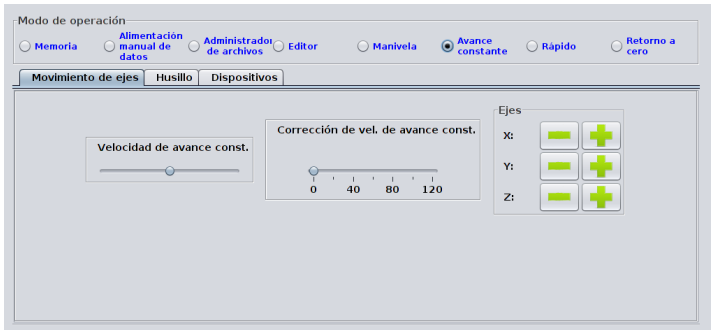
Crea un objeto de tipo MdiMode que representa al modo de introducción manual de datos. Se establece el nombre clave y valor del comando del modo de operación.

IV.18.2 Métodos públicos

setDescriptionViewPanel public void setDescriptionViewPanel()

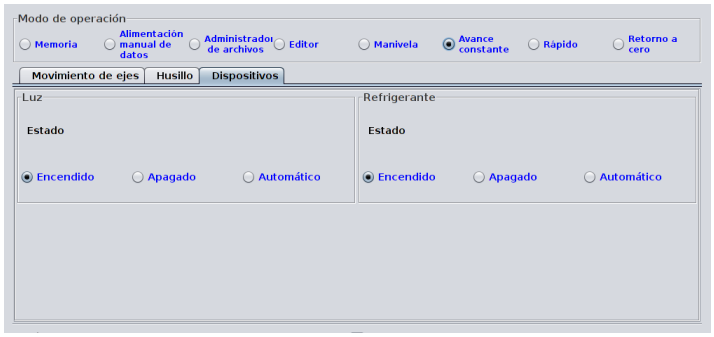
Propuesto por Mode, establece los componentes cuyos identificadores son:

Id.ATP
Id.SPINDLE
Id.DEVICES
Id.EMERGENCY_STOP
Id.OPERATION_MODE
Id.MOTION_TYPE
Id.CARTESIAN_PLANE
Id.DIMENSIONING_MODE
Id.CUTTING_MODE
Id.SPINDLE_DIRECTION



(a) Movimiento de los ejes

(b) Husillo



(c) Dispositivos

Figura 4.31: Capturas de la lámina inferior del modo de avance constante.

Id.SPINDLE_MOTION
Id.AXIS_LIMIT_SWITCH
Id.AXIS_CURRENT_POSITION
Id.TOOL_MAGAZINE_CAPACITY
Id.MAXIMUM_TOOL_INDEX
Id.TOOL_SELECTION_TYPE
Id.SELECTED_TOOL_INDEX
Id.NEXT_TOOL_INDEX
Id.AXIS_MOTION_TRAVEL_RANGE
Id.AXIS_REFERENCE_POINT_
RETURN_DIRECTION

La lámina superior resultante se muestra en la figura 4.32.

setDescriptionEditablePanel public void setDescriptionEditablePanel()
Propuesto por Mode, establece los componentes cuyos identificadores son:

Figura 4.32: Captura de la lámina superior del modo de introducción manual de datos.

Id.BLOCK_EXECUTION
Id.OPTIONAL_STOP
Id.BLOCK_SKIP
Id.DRY_RUN
Id.MST_LOCK
Id.XY_AXES_LOCK
Id.Z_AXIS_NEGLECT
Id.CUTTING_FEEDRATE_OVERRIDE
Id.SPINDLE_SPEED_OVERRIDE
Id.MACHINING_CYCLE_STATUS
Id.RAPID_TRAVERSE_RATE_
OVERRIDE
AnotherComponent.MDI_BUFFER

La lámina inferior resultante, y sus solapas, se muestran en la figura 4.32.

IV.19. Clase FileManagerMode

public class FileManagerMode extends Mode

La clase FileManagerMode corresponde al modo de administración de archivos. Extiende a la clase Mode.

IV.19.1 Constructor

FileManagerMode public FileManagerMode()

Crea un objeto de tipo FileManagerMode que representa al modo de administración de archivos. Se establece el nombre clave y valor del comando del modo de operación.

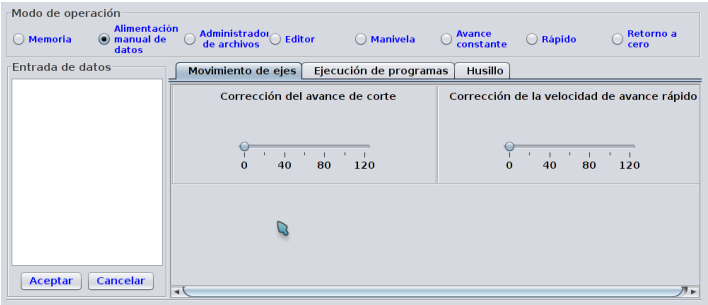
IV.19.2 Métodos públicos

setDescriptionViewPanel public void setDescriptionViewPanel()

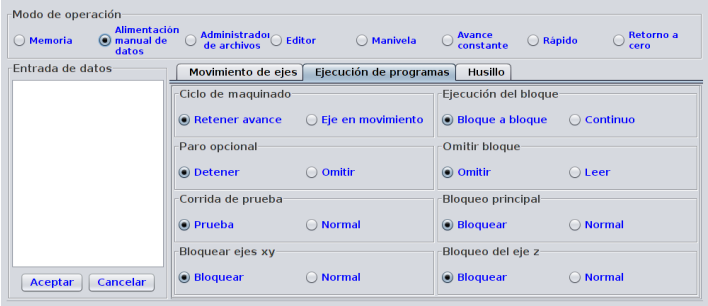
Propuesto por Mode, establece los componentes cuyos identificadores son:

Id.OPERATION_MODE
AnotherComponent.FILE_MANAGER

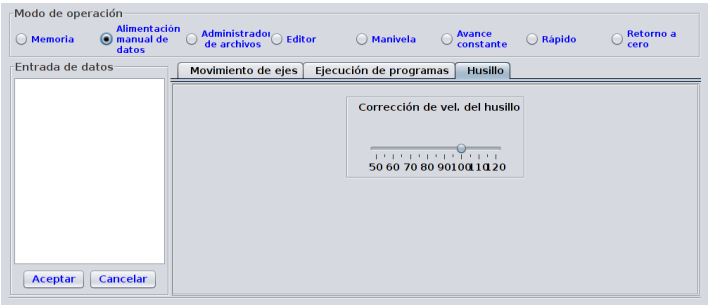
La lámina superior resultante se muestra en la figura 4.34.



(a) Movimiento de los ejes



(b) Ejecución del programa



(c) Husillo

Figura 4.33: Capturas de la lámina inferior del modo de introducción manual de datos.

setDescriptionEditablePanel public void setDescriptionEditablePanel()
Propuesto por Mode, establece los componentes encargados de las tareas de administración, cuyo identificador es AnotherComponent.TASKBAR_FILE_MANAGER. La lámina inferior resultante se muestran en la figura 4.35.

IV.20. Clase EditorMode

public class EditorMode extends Mode

La clase EditorMode corresponde al modo de operación editor. Extiende a la clase Mode.

Figura 4.34: Captura de la lámina superior del modo de administración de archivos.

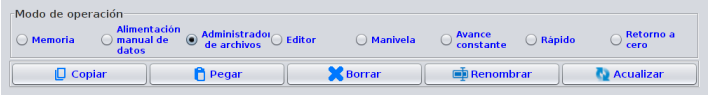


Figura 4.35: Captura de la lámina inferior del modo de administración de archivos.

IV.20.1 Constructor

EditorMode public EditorMode()

Crea un objeto de tipo EditorMode que representa al modo editor. Se establece el nombre clave y valor del comando del modo de operación.

IV.20.2 Métodos públicos

setDescriptionViewPanel public void setDescriptionViewPanel()

Propuesto por Mode, establece los componentes cuyos identificadores son:

Id.OPERATION_MODE
AnotherComponent.EDITOR_AREA

La lámina superior resultante se muestra en la figura 4.36.

setDescriptionEditablePanel public void setDescriptionEditablePanel()

Propuesto por Mode, establece los componentes cuyos identificadores son:

Id.SPINDLE
Id.CUTTING_DATA_EDITOR
Id.JOG_DATA_EDITOR
Id.RTR_DATA_EDITOR
Id.ZRTR_DATA_EDITOR
Id.AXIS_MOTION_TRAVEL_RANGE
Id.TOOL_MAGAZINE_CAPACITY
Id.MAXIMUM_TOOL_INDEX
Id.TOOL_SELECTION_TYPE
Id.SELECTED_TOOL_INDEX
Id.TOOL_HOLDER_DATA_EDITOR
Id.DEFAULT_METRIC_SYSTEM

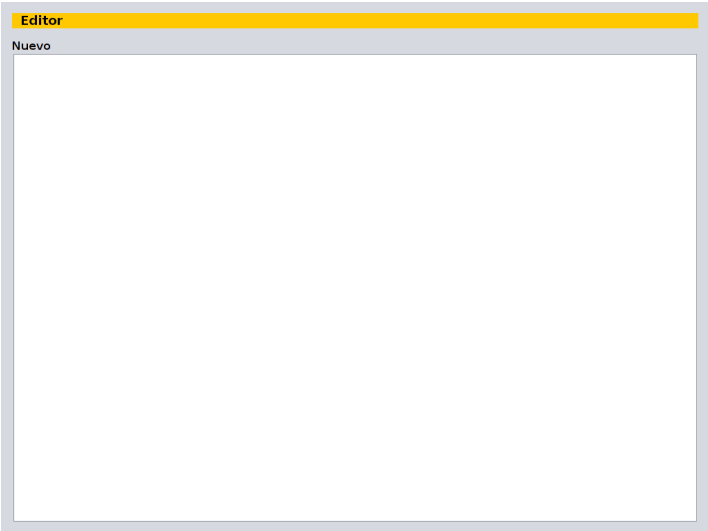


Figura 4.36: Captura de la lámina superior del modo editor.

Id.DEFAULT_DIMENSIONING_MODE	AnotherComponent.TASKBAR
Id.AXIS_REFERENCE_POINT_ RETURN_DIRECTION	AnotherComponent.VALIDATE_ CHANGE

La lámina inferior resultante, y sus solapas, se muestran en la figura 4.37.

IV.21. Clase RapidMode

public class RapidMode extends Mode

La clase RapidMode corresponde al modo de operación de avance rápido. Extiende a la clase Mode.

IV.21.1 Constructor

JogMode public RapidMode()

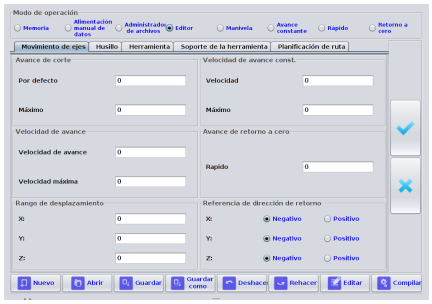
Crea un objeto de tipo RapidMode que representa al modo de avance rápido. Se establece el nombre clave y valor del comando del modo de operación.

IV.21.2 Métodos públicos

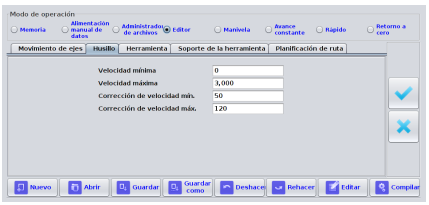
setDescriptionViewPanel public void setDescriptionViewPanel()

Propuesto por Mode, establece los componentes cuyos identificadores son:

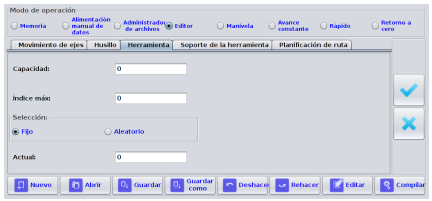
Id.EMERGENCY_STOP	Id.AXIS_MOTION_TRAVEL_RANGE
Id.OPERATION_MODE	Id.SPINDLE
Id.ATP	Id.DEVICES
Id.AXIS_LIMIT_SWITCH	Id.FEEDRATE
Id.AXIS_CURRENT_POSITION	



(a) Movimiento de los ejes

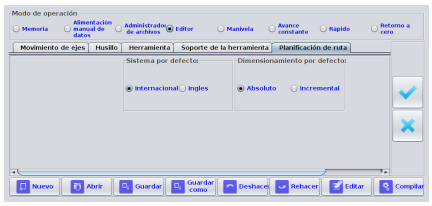


(b) Husillo



(c) Herramienta

(d) Soporte de la herramienta



(e) Planificación de ruta

Figura 4.37: Capturas de la lámina inferior del modo editor.

La lámina superior resultante se muestra en la figura 4.38.

setDescriptionEditablePanel public void setDescriptionEditablePanel()
Propuesto por Mode, establece los componentes cuyos identificadores son:

Id.RAPID_TRAVERSE_RATE_OVERRIDE
Id.SPINDLE_DIRECTION
Id.SPINDLE_MOTION
Id.SPINDLE_SPEED_OVERRIDE
Id.DEVICES

La lámina inferior resultante, y sus solapas, se muestran en la figura 4.39.

IV.22. Clase MemoryMode

public class MemoryMode extends Mode

La clase MemoryMode corresponde al modo de operación memoria. Extiende a la clase Mode.

Figura 4.38: Captura de la lámina superior del modo de avance rápido.

IV.22.1 Constructor

MemoryMode public MemoryMode()

Crea un objeto de tipo MemoryMode que representa al modo memoria. Se establece el nombre clave y valor del comando del modo de operación.

IV.22.2 Métodos públicos

setDescriptionViewPanel public void setDescriptionViewPanel()

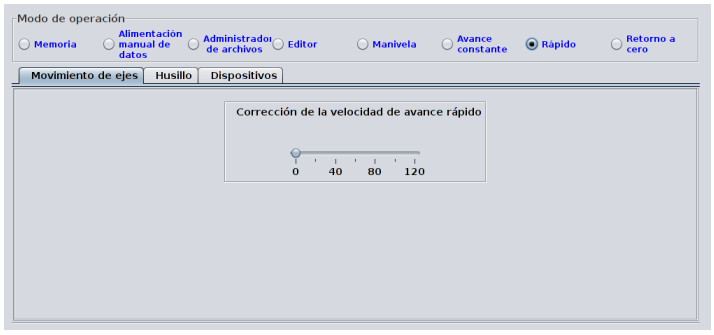
Propuesto por Mode, establece los componentes cuyos identificadores son:

Id.AXIS_CURRENT_POSITION
Id.MOTION_TYPE
Id.CARTESIAN_PLANE
Id.DIMENSIONING_MODE
Id.CUTTING_MODE
Id.SPINDLE_DIRECTION
Id.SPINDLE_MOTION
Id.ATP
Id.PROGRAM_ZERO
Id.SPINDLE
Id.FEEDRATE
Id.AXIS_LIMIT_SWITCH
Id.TOOL_MAGAZINE_CAPACITY
Id.MAXIMUM_TOOL_INDEX
Id.TOOL_SELECTION_TYPE
Id.SELECTED_TOOL_INDEX
Id.NEXT_TOOL_INDE
Id.DEVICES
Id.OPERATION_MODE
Id.EMERGENCY_STOP
AnotherComponent.PROGRAM_
EXECUTION_AREA

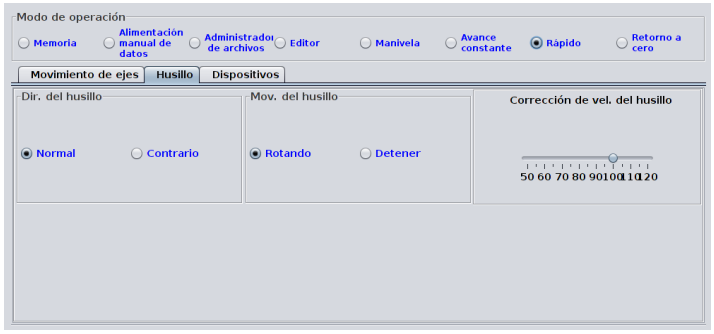
La lámina superior resultante se muestra en la figura 4.40.

setDescriptionEditablePanel public void setDescriptionEditablePanel()

Propuesto por Mode, establece los componentes cuyos identificadores son:



(a) Movimiento de los ejes



(b) Husillo

(c) Dispositivos

Figura 4.39: Capturas de la lámina inferior del modo de avance rápido.

Id.CUTTING_FEEDRATE_OVERRIDE	Id.BLOCK_SKIP
Id.RAPID_TRAVERSE_RATE_	Id.DRY_RUN
OVERRIDE	Id.MST_LOCK
Id.MACHINING_CYCLE_STATUS	Id.XY_AXES_LOCK
Id.BLOCK_EXECUTION	Id.Z_AXIS_NEGLECT
Id.OPTIONAL_STOP	Id.SPINDLE_SPEED_OVERRIDE

La lámina inferior resultante, y sus solapas, se muestran en la figura 4.41.

Figura 4.40: Captura de la lámina superior del modo memoria.

IV.23. Clase HandwheelMode

public class HandwheelMode extends Mode

La clase HandwheelMode corresponde al modo de operación de la manivela. Extiende a la clase Mode.

IV.23.1 Constructor

MemoryMode public HandwheelMode()

Crea un objeto de tipo HandwheelMode que representa al modo de la manivela. Se establece el nombre clave y valor del comando del modo de operación.

IV.23.2 Métodos públicos

setDescriptionViewPanel public void setDescriptionViewPanel()

Propuesto por Mode, establece los componentes cuyos identificadores son:

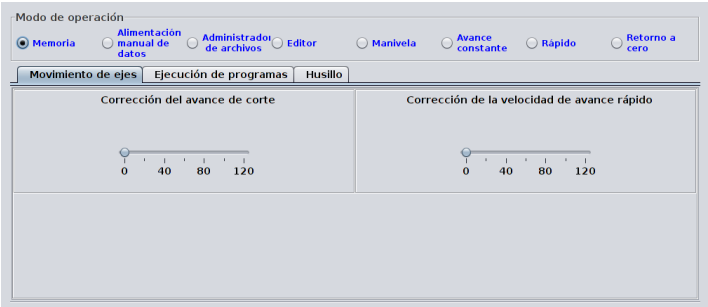
Id.OPERATION_MODE	Id.AXIS_CURRENT_POSITION
Id.EMERGENCY_STOP	Id.AXIS_MOTION_TRAVEL_RANGE
Id.ATP	Id.SPINDLE
Id.AXIS_LIMIT_SWITCH	Id.DEVICES

La lámina superior resultante se muestra en la figura 4.42.

setDescriptionEditablePanel public void setDescriptionEditablePanel()

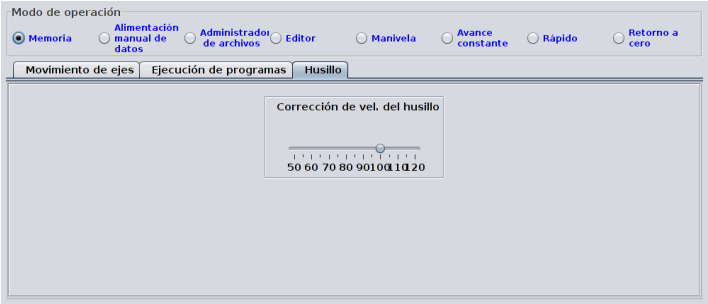
Propuesto por Mode, establece los componentes cuyos identificadores son:

Id.WORK_OFFSET	Id.HANDWHEEL_AXIS
Id.TOOL_OFFSET	Id.SPINDLE_DIRECTION
Id.HANDWHEEL_FACTOR	Id.SPINDLE_SPEED_OVERRIDE



(a) Movimiento de los ejes

(b) Ejecución de programas



(c) Husillo

Figura 4.41: Capturas de la lámina inferior del modo memoria.

Id.SPINDLE_SPEED	Id.DEVICES AnotherComponent.VALIDATE_ CHANGE

La lámina inferior resultante, y sus solapas, se muestran en la figura 4.43.

IV.24. Clase InfoManager

public abstract class InfoManager

La clase InfoManager proporciona métodos encargados de administrar la información entre la GUI y los módulos restantes. Los métodos públicos de clase son usados dentro de cualquier código de la GUI que los necesite, los métodos de objeto son usados por el código nativo.

Figura 4.42: Captura de la lámina superior del modo de operación de la manivela.

IV.24.1 Constructor

InfoManager public InfoManager()

Crea un nuevo objeto de tipo InfoManager encargado de administrar la información entre el código nativo y la interfaz gráfica de usuario. Inicializa los ejemplares de los modos de operación y datos.

IV.24.2 Métodos públicos

initData Las variantes son:

- public boolean initData(int id, int value)
- public boolean initData(int id, long value)

Recibe los valores iniciales de los datos en parejas formadas por el identificador del dato y su correspondiente valor. Los valores son comunicados al ejemplar de la clase Data para ser dirigidos a la categoría correspondiente.

Parámetros:

`id` - identificador del dato, definido en la interfaz KbdCmdId o DataConst.

`value` - el valor que se asignara al dato.

initDataWorkOffset public void initDataWorkOffset(int x, int y, int z)

Recibe los datos iniciales de las compensaciones del área de trabajo.

Parámetros:

`x` - compensación de trabajo del eje x.

`y` - compensación de trabajo del eje y.

`z` - compensación de trabajo del eje z.

(a) Planificación de la ruta

(b) Movimiento de los ejes

(c) Husillo

(d) Dispositivos

Figura 4.43: Capturas de la lámina inferior del modo de operación de la manivela.

initDataToolOffset public void initDataToolOffset(int lenght, int diameter)

Recibe los datos iniciales de las compensaciones de las herramientas presentes.

Parámetros:

`lenght` - compensación de la herramienta correspondiente a su longitud.

`diameter` - compensación de la herramienta correspondiente a su diámetro.

getCommand public long[] getCommand()

Proporciona los comandos existentes en la lista `commandSend`, los cuales son generados por los controles del teclado de la interfaz gráfica. El arreglo de enteros proporcionado esta formado por el identificador del comando en la primera posición, seguido por el valor.

getDataValue public long[] getDataValue()

Proporciona los valores existentes en la lista `sendingDataValue`, que son generados por los campos editables y que no pertenecen al teclado de la máquina. Los valores proporcionados se dividen en tres casos.

Caso 1. Si el valor del primer elemento del arreglo corresponde al de la constante `DataConst.WORK_OFFSET`, el arreglo constara de cuatro enteros más, siendo el índice de la compensación de trabajo seguido de los nuevos valores del eje x, eje y y eje z.

Caso 2. Si el valor del primer elemento del arreglo corresponde al de la constante `DataConst.TOOL_OFFSET`, el arreglo constara de tres enteros más, siendo el índice de la herramienta seguido de los nuevos valores para la longitud y el diámetro.

Caso 3. En otro caso es una dupla de enteros, correspondiendo al identificador y el valor modificado del dato que representan.

setCommand Las variantes son:

- public void setCommand(int id, int value)
- public void setCommand(int id, long value)

Comunica a la GUI cambios generados por el teclado a los datos correspondientes.

Parámetros:

`id` - identificador del comando de teclado, definido en la clase KbdCmdId.

`value` - nuevo valor.

setDataValue Las variantes son:

- public void setDataValue(int id,int value)
- public void setDataValue(int id,long value)

Comunica a la GUI cambios generados en datos no presentes en los comandos del teclado. Como por ejemplo el cambio en las coordenadas de la posición actual.

Parámetros:

`id` - identificador del dato, definido en DataConst.

`value` - nuevo valor del dato.

changeExists public boolean changeExists()
Informa si existen cambios generados por los controles del teclado. Retorna verdadero (true) si existen cambios en la lista `commandSend` y falso (false) si no hay cambios.

existChangedData public boolean existChangedData()
Informa si se han editado valores de datos ajenos a los comandos del teclado. Retorna verdadero (true) si existen cambios en la lista `sendingDataValue` y falso (false) si no hay cambios.

newTasksExists public boolean newTasksExists()
Informa si existen tareas pendientes relativas a la edición y administración de archivos. Retorna verdadero (true) si existen tareas en la lista `taskSend` y falso (false) si no.

addCommand Las variantes son:

- public static void addCommand(int id, int value)
- public static void addCommand(int id, long value)

Agrega en la lista `commandSend` comandos generados por los controles de la GUI.

Parámetros:

`id` - identificador del comando de teclado, definido en la clase KbdCmdId.

`value` - nuevo valor.

addDataValue Las variantes son:

- public static void addDataValue(int id, int value)
- public static void addDataValue(int id, long value)

Agrega a la lista `sendingDataValue` cambios realizados a los datos, a través de la interfaz gráfica de usuario.

Parámetros:

`id` - identificador del dato, definido en DataConst.

`value` - nuevo valor del dato.

addTask public static void addTask(int value)
Agrega a la cola tareas pendientes de edición y administración de archivos.

Parámetros:

`value` - valor correspondiente al identificador de la tarea.

addWorkOffsets public static void addWorkOffsets(int workOffsetIndex, int indexAxis, int value)
Agrega a la lista de datos únicamente cambios correspondientes al desplazamiento del área de trabajo.

Parámetros:

`workOffsetIndex` - índice de la compensación de trabajo correspondiente.

`indexAxis` - índice del eje al que corresponde el valor, es decir, para el eje X es 0, eje Y es 1 y eje Z es 2.

`value` - nuevo valor de desplazamiento.

addToolOffsets public static void addToolOffsets(int toolOffsetIndex, int lenght_radius, int value)
Agrega a la lista de datos únicamente cambios en las compensaciones de las herramienta.

Parámetros:

`toolOffsetIndex` - índice de la herramienta.

`lenght_radius` - índice para determinar si el valor corresponde a la longitud o al radio.

`value` - nuevo radio o longitud.

getTask public int getTask()
Proporciona el identificador de tareas de edición pendientes en la lista `taskSend`, uno a uno con cada llamada al método.

browserRefresh public void browserRefresh()
Actualiza el administrador de archivos.

getFileName public String getFileName()

Proporciona una cadena de texto con el nombre del archivo para realizar la ultima tarea de edición y administración en proceso (Abrir, copiar, copiar como, renombrar, borrar).

getNewFileName public String getNewFileName()

Proporciona el nuevo nombre para un archivo en la tarea del administrador "Renombrar".

setLineText public void setLineText(String line)

Agrega una nueva línea de texto para ser visualizada en alguna área de texto de la GUI, dependiendo del modo de operación.

getLineText public String getLineText()

Proporciona, una a una, líneas de texto del editor o `null` si se llega al final del texto u otro caso.

addElement public void addElement(String name_,int size_,String date_)

Agrega información de los archivos de un directorio. Existen tres listas que almacenan cada uno de los datos proporcionados al método, que son `name`, `size` y `date`.

Parámetros:

`name_` - nombre del archivo.

`size_` - tamaño del archivo.

`date_` - fecha de modificación del archivo.

getName public static String[] getName()

Proporciona un arreglo de cadenas de texto con los nombres agregados a través del metodo `addElement`. La lista `name` es vaciada en este método.

getSize public static Integer[] getSize()

Proporciona un arreglo de objetos de tipo Integer con los valores agregados a través del metodo `addElement`. La lista `size` es vaciada en este método.

getDate public static String[] getDate()

Proporciona un arreglo de cadenas de texto con las fechas de modificación agregadas a través del metodo `addElement`. La lista `date` es vaciada en este método.

selectForm public void selectForm(int value)

Establece el modo de operación solicitado.

Parámetros:

`value` - identificador del modo de operación.

getMode public Mode getMode(int value)

Proporciona un ejemplar de un modo de operación.

Parámetros:

`value` - identificador del modo de operación.

IV.24.3 Métodos privados

removeCommand private static long[] removeCommand()
Proporciona y borra un elemento de la lista `commandSend`.

IV.24.4 Campos privados

sendingDataValue private static ArrayList<long[]>sendingDataValue
Lista de arreglos de enteros, almacena los cambios realizados en datos.

commandSend private static ArrayList<long[]>commandSend
Lista de arreglos de enteros, almacena los cambios realizados por controles a datos.

commandReceived private static ArrayList<long[]>commandReceived
Lista de arreglos de enteros, almacena cambios proporcionados por el código nativo.

taskSend private static ArrayList<Integer>taskSend
Lista de tipo Integer, almacena identificadores de tareas pendientes de edición y administración de archivos.

name private static ArrayList<String>name
Lista de cadenas de texto, almacena nombres de archivos.

size private static ArrayList<Integer>size
Lista de cadena de texto, almacena el tamaño de los archivos.

date private static ArrayList<String>date
Lista de cadenas de texto, almacena la fecha de modificación de los archivos.

editorMode private EditorMode editorMode
Ejemplar de la clase EditorMode, correspondiente al modo editor.

memoryMode private MemoryMode memoryMode
Ejemplar de la clase MemoryMode, correspondiente al modo memoria.

mdiMode private MdiMode mdiMode
Ejemplar de la clase MdiMode, correspondiente al modo de introducción manual de datos.

fileManagerMode private FileManagerMode fileManagerMode
Ejemplar de la clase FileManagerMode, correspondiente al modo administrador de archivos.

handwheelMode private HandwheelMode handwheelMode
Ejemplar de la clase HandwheelMode, corespondiente al modo manivela.

jogMode private JogMode jogMode
Ejemplar de la clase jogMode, correspondiente al modo avance constante.

rapidMode private RapidMode rapidMode
Ejemplar de la clase RapidMode, correspondiente al modo avance rápido.

zeroReturnMode private ZeroReturnMode zeroReturnMode
Ejemplar de clase ZeroReturnMode, correspondiente al modo retorno a cero.

xyz private static ArrayList<long[]>xyz
Lista de arreglos de enteros, almacena las correcciones del area de trabajo.

dimensions private static ArrayList<long[]>dimensions
Lista de arreglos de enteros, almacena las correcciones de las herramientas.

data private Data data
Ejemplar de clase Data, la cual engloba todas las categorías de datos.

IV.25. Clase CoreGui

public class CoreGui extends InfoManager

La clase CoreGui inicializa y administra los modos de operación (características heredadas de InfoManager), asi como los marcos donde se colocaran todos los componentes de la interfaz gráfica de usuario y el teclado. También proporciona funciones importantes para la interacción de la GUI con el código nativo.

Cuenta con dos marcos, uno superior y otro inferior. El marco superior es el que se coloca en la pantalla no táctil y contiene por lo general componentes no editables pensados solamente para visualizar información. Por otro lado, el marco inferior se coloca en la pantalla táctil y contiene el teclado virtual, los controles de la máquina, campos editables y el menú de modo de operación.

Los dos marcos tiene un tamaño inicial de 1024x768, siendo este el tamaño mínimo para visualizar correctamente las pantallas. Además, las pantallas se maximizan y adaptan a resoluciones mayores.

IV.25.1 Constructor

CoreGui public CoreGui()
Crea un nuevo objeto de tipo CoreGui, inicializa los modos de operación y los marcos superior e inferior que corresponden a la pantalla normal y la pantalla táctil.

IV.25.2 Métodos privados

initializeTopFrame private void initializeTopFrame()
Inicializa el marco superior.

initializeBottomFrame private void initializeBottomFrame()
Inicializa el marco inferior. Puede producir una excepción si la configuración de la plataforma no permite el control de entradas de bajo nivel, que es necesario para crear el teclado virtual.[20]

getModesPanel private JPanel getModesPanel()
Proporciona un menú formado por una lámina que contiene botones de radio para elegir el modo de operación.

[20] Véase la clase Robot

IV.25.3 Campos privados

radioEdit private JRadioButton radioEdit
Botón de radio correspondiente al modo editor.

radioMemory private JRadioButton radioMemory
Botón de radio correspondiente al modo memoria.

radioMdi private JRadioButton radioMdi
Botón de radio correspondiente al modo de introducción manual de datos.

radioFileManager private JRadioButton radioFileManager
Botón de radio correspondiente al modo de administrador de archivos.

radioHandwheel private JRadioButton radioHandwheel
Botón de radio correspondiente al modo de manivela.

radioJog private JRadioButton radioJog
Botón de radio correspondiente al modo de avance constante.

radioRapid private JRadioButton radioRapid
Botón de radio correspondiente al modo de avance rápido.

radioZeroReturn private JRadioButton radioZeroReturn
Botón de radio correspondiente al modo de retorno a cero.

topFrame private JFrame topFrame
Marco superior de la interfaz gráfica de usuario.

bottomFrame private JFrame bottomFrame
Marco inferior de la interfaz gráfica de usuario.

WIDTH_FINAL private static final int WIDTH_FINAL
Campo constante correspondiente al ancho inicial de los marcos.

HEIGHT_FINAL private static final int HEIGHT_FINAL
Campo constante correspondiente a la altura inicial de los marcos.

keyboard private Keyboard keyboard
Ejemplar del teclado virtual.

IV.26. Interfaz KbdCmdId

public interface KbdCmdId

La interfaz KbdCmdId contiene definidos los identificadores y valores correspondientes a eventos de un teclado de fresadora. Toda acción del teclado comunica al código nativo y a la GUI una pareja formada por el identificador del comando, definido en la clase interna CmdId, y el valor correspondiente que, dependiendo del comando, puede ser uno de los definidos en la clase interna CmdValue o un valor generado por el mismo teclado.

IV.26.1 Clases internas

CmdId public class CmdId

Agrupa constantes enteras para identificar comandos del teclado de una fresadora. Toda las constantes son declaradas públicas y estáticas.

Campos constantes

CHARACTER Identificador de los comandos generados por teclas de caracteres. Valor: 0x00.

KEYSTROKE Identificador de los comandos generados por teclas como Control o Alt. Valor: 0x01.

EMERGENCY Identificador de los comandos generados por el botón de paro de emergencia. Valor: 0x02.

MODE Identificador de los comandos generados por el menú de selección de modo de operación. Valor: 0x04.

CYCLE Identificador de los comandos generados por los controles de estado del ciclo de maquinado. Valor: 0x03.

OPTION Identificador de los comandos generados por los controles del comportamiento de la ejecución de programas. Valor: 0x05.

FEEDRATE_OVERRIDE Identificador de los comandos generados por los controles de corrección de la velocidad de avance. Valor: 0x06.

JOG_FEEDRATE Identificador de los comandos generados por los controles de la velocidad de avance constante. Valor: 0x07.

RAPID_FEEDRATE_OVERRIDE Identificador de los comandos generados por los controles de la corrección de la velocidad de avance rápido. Valor: 0x08.

AXIS Identificador de los comandos generados por los controles de movimiento de los ejes del modo de avance contante. Valor: 0x09.

SPINDLE Identificador de los comandos generados por los controles de movimiento del husillo. Valor: 0x0A.

SPINDLE_SPEED Identificador de los comandos generados por el control de la velocidad del husillo. Valor: 0x0B

SPINDLE_OVERRIDE Identificador de los comandos generados por el control de la corrección de la velocidad del husillo. Valor: 0x0C.

HANDWHEEL Identificador de los comandos generados por los controles de opciones del factor de avance y elección de eje de la manivela. Valor: 0x0D.

TOOL Identificador de los comandos generados por los controles de elección de herramienta. Valor: 0x0E.

DEVICE Identificador de los comandos generados por los controles de estado de los dispositivos. Valor: 0x0F.

CmdValue public class CmdValue

Agrupa constantes para los valores que pueden tomar algunos de los comandos del teclado de una fresadora. Toda las constantes son declaradas públicas y estáticas.

Campos públicos constantes

Posibles valores del comando “EMERGENCY”.

EMERGENCY_STOP Paro de emergencia activado. Valor 0x00.

EMERGENCY_CONTINUE Paro de emergencia desactivado. Valor 0x01.

Posibles valores del comando “MODE”. Corresponden a los ocho modos de operación.

MODE_MEMORY Modo memoria. Valor: 0x00.

MODE_MDI Modo de introducción manual de datos. Valor: 0x01.

MODE_FILE_MANAGER Modo de administrador de archivos. Valor: 0x02.

MODE_EDITOR Modo editor. Valor: 0x03.

MODE_HANDWHEEL Modo manivela. Valor: 0x04.

MODE_JOG Modo de avance constante. Valor: 0x05.

MODE_RAPID Modo de avance rápido. Valor: 0x06.

MODE_HOME Modo de retorno a cero. Valor: 0x07.

Posibles valore del comando “HANDWHEEL”. Corresponden al factor del incremento de la manivela y los tres ejes.

HANDWHEEL_FACTOR_1 Valor: 0x02.

HANDWHEEL_FACTOR_10 Valor: 0x03.

HANDWHEEL_FACTOR_100 Valor: 0x04.

HANDWHEEL_SELECT_AXIS_X Valor: 0x05.

HANDWHEEL_SELECT_AXIS_Y Valor: 0x06.

HANDWHEEL_SELECT_AXIS_Z Valor: 0x07.

Posibles valores del comando “CYCLE”.

CYCLE_START Ejes en movimiento. Valor: 0x00.

CYCLE_FEED_HOLD Retener ciclo de maquinado. Valor: 0x01.

Posibles valores del comando “OPTION”.

OPTION_SINGLE_BLOCK_OFF Valor: 0x00.

OPTION_SINGLE_BLOCK_ON Valor: 0x01.

OPTION_OPTIONAL_STOP_OFF Valor: 0x02.

OPTION_OPTIONAL_STOP_ON Valor: 0x03.

OPTION_BLOCK_SKIP_OFF Valor: 0x04.

OPTION_BLOCK_SKIP_ON Valor: 0x05.

OPTION_DRY_RUN_OFF Valor: 0x06.

OPTION_DRY_RUN_ON Valor: 0x07.

OPTION_MST_LOCK_OFF Valor: 0x08.

OPTION_MST_LOCK_ON Valor: 0x09.

OPTION_MACHINE_LOCK_OFF Valor: 0x0A.

OPTION_MACHINE_LOCK_ON Valor: 0x0B.

Posibles valores del comando “AXIS”.

AXIS_X_POSITIVE_IGNORE Valor: 0x00.

AXIS_X_POSITIVE_SELECT Valor: 0x01.

AXIS_X_NEGATIVE_IGNORE Valor: 0x02.

AXIS_X_NEGATIVE_SELECT Valor: 0x03.

AXIS_Y_POSITIVE_IGNORE Valor: 0x10.

AXIS_Y_POSITIVE_SELECT Valor: 0x11.

AXIS_Y_NEGATIVE_IGNORE Valor: 0x12.

AXIS_Y_NEGATIVE_SELECT Valor: 0x13.

AXIS_Z_POSITIVE_IGNORE Valor: 0x20.

AXIS_Z_POSITIVE_SELECT Valor: 0x21.

AXIS_Z_NEGATIVE_IGNORE Valor: 0x22.

AXIS_Z_NEGATIVE_SELECT Valor: 0x23.

Posibles valores del comando “SPINDLE”.

SPINDLE_NORMAL_ROTATION Rotación del husillo en dirección a las manecillas del reloj. Valor: 0x00.

SPINDLE_REVERSE_ROTATION Rotación del husillo en contra de las manecillas del reloj. Valor: 0x01.

SPINDLE_START Rotar. Valor: 0x02.

SPINDLE_STOP Parar. Valor: 0x03.

Posibles valores del comando “DEVICE”.

DEVICE_DISABLE Desactivado. Valor: 0x00.

DEVICE_ENABLE Activado. Valor: 0x01.

DEVICE_AUTOMATIC Automatico. Valor: 0x02.

IV.27. Interfaz DataConst

public interface DataConst

La interfaz DataConst contiene definidos los identificadores correspondientes a los distintos datos generados por el sistema de una fresadora, identificadores para las distintas categorías de datos e identificadores de utilidad para el armado de las pantallas de los modos de operación. Estos últimos, correspondientes a la enumeración `Id`, sirven para hacer referencia a los componentes que forman cada una de las pantallas de los modos de operación, entonces en cada modo de operación basta con listar que componentes deben aparecer y en donde colocarlos.[21]

IV.27.1 Enumeraciones

Category public static enum Category

Enumeración de las categorías de datos.

MACHINE_OPERATION Operación de la máquina.

PATH_PLANNING Planificación de la ruta.

AXIS_MOTION Movimiento de los ejes.

PROGRAM_EXECUTION Ejecución de programas.

SPINDLE Husillo.

ATC Cambio automático de herramienta.

DEVICES Dispositivos.

TOOL_HOLDER Soporte de la herramienta.

OTHER Otro.

Id public static enum Id

Identificador de los componentes donde se visualizan o editan los datos. Cuenta con un constructor, un método y un campo para establecer, proporcionar y almacenar la categoría a la que pertenecen.

Categoría: MACHINE_OPERATION.

EMERGENCY_STOP
OPERATION_MODE

Categoría: PATH_PLANNING.

MOTION_TYPE
CARTESIAN_PLANE
METRIC_SYSTEM
DIMENSIONING_MODE
ATP
PROGRAM_ZERO
DEFAULT_METRIC_SYSTEM
DEFAULT_DIMENSIONING_MODE
WORK_OFFSET
TOOL_OFFSET

[21] Véase la clase DataCategory y sus extensiones.

Categoría: AXIS_MOTION.

CUTTING_MODE
FEEDRATE
AXIS_LIMIT_SWITCH
AXIS_CURRENT_POSITION
AXIS_MOTION_TRAVEL_RANGE
ZERO_RETURN
CUTTING_FEEDRATE_OVERRIDE
JOG_FEEDRATE
JOG_FEEDRATE_OVERRIDE
HANDWHEEL_FACTOR
HANDWHEEL_AXIS
CUTTING_DATA_EDITOR
JOG_DATA_EDITOR
RTR_DATA_EDITOR
ZRTR_DATA_EDITOR
JOG_AXES
DEFAULT_CUTTING_FEEDRATE
MAXIMUM_CUTTING_FEEDRATE
MAXIMUM_JOG_FEEDRATE
RAPID_TRAVERSE_RATE
FAST_ZRTR
MAXIMUM_TRAVERSE_RATE
X_AXIS_MOTION_TRAVEL_RANGE
Y_AXIS_MOTION_TRAVEL_RANGE
Z_AXIS_MOTION_TRAVEL_RANGE
RAPID_TRAVERSE_RATE_
OVERRIDE
AXIS_REFERENCE_POINT_
RETURN_DIRECTION
X_AXIS_REFERENCE_POINT_
RETURN_DIRECTION
Y_AXIS_REFERENCE_POINT_
RETURN_DIRECTION
Z_AXIS_REFERENCE_POINT_
RETURN_DIRECTION

Categoría: PROGRAM_EXECUTION

JOB_NAME
BLOCK_EXECUTION
OPTIONAL_STOP
BLOCK_SKIP
DRY_RUN
MST_LOCK
XY_AXES_LOCK
Z_AXIS_NEGLECT
MACHINING_CYCLE_
STATUS

Categoría: SPINDLE.

SPINDLE
SPINDLE_MOTION
SPINDLE_DIRECTION
SPINDLE_SPEED_OVERRIDE
SPINDLE_SPEED
MINIMUM_SPINDLE_SPEED
MAXIMUM_SPINDLE_SPEED
MINIMUM_SPINDLE_SPEED_
OVERRIDE
MAXIMUM_SPINDLE_SPEED_
OVERRIDE

Categoría: ATC.

TOOL_DATA_EDITOR
TOOL_MAGAZINE_CAPACITY
MAXIMUM_TOOL_INDEX
TOOL_SELECTION_TYPE
SELECTED_TOOL_INDEX
NEXT_TOOL_INDEX

Categoría: DEVICES.

DEVICES
NUMBER_OF_DEVICES

Categoría: TOOL_HOLDER.

TOOL_HOLDER_DATA_EDITOR
MAXIMUM_TOOL_DIAMETER
MAXIMUM_TOOL_LENGTH

Categoría: OTHER.

OTHER

IV.27.2 Campos públicos

Los siguientes campos se definen como constantes.

MOTION_TYPE Tipo de movimiento. Valor = 0x10.

CARTESIAN_PLANE Plano cartesiano. Valor = 0x11.

DEFAULT_METRIC_SYSTEM Sistema métrico por omisión. Valor = 0x4E.

METRIC_SYSTEM Sistema métrico. Valor = 0x12.

DIMENSIONING_MODE Modo de dimensionamiento. Valor = 0x13.

DEFAULT_DIMENSIONING_MODE Modo de dimensionamiento por omisión. Valor = 0x50.

ATP_X Valor del eje x de la posición actual. Valor = 0x16.

ATP_Y Valor del eje y de la posición actual. Valor = 0x17.

ATP_Z Valor del eje z de la posición actual. Valor = 0x18.

ACTIVE_WORK_OFFSET_INDEX Índice de la compensación del área de trabajo activa. Valor = 0x19.

ACTIVE_TOOL_OFFSET_INDEX Índice de la compensación de la herramienta activa. Valor = 0x1A.

NUM_WORK_OFFSETS Número total de compensaciones de área de trabajo. Valor = 0x4F.

MAXIMUM_CUTTING_FEEDRATE Velocidad máxima de corte. Valor = 0x35.

DEFAULT_CUTTING_FEEDRATE Velocidad de avance de corte por omisión. Valor = 0x34.

CUTTING_MODE Modo de corte. Valor = 0x14.

CUTTING_FEEDRATE Velocidad de avance de corte. Valor = 0x1B.

X_LEFT_AXIS_HARD_LIMIT_SWITCH Interruptor límite izquierdo del eje X. Valor = 0x2B.

X_RIGHT_AXIS_HARD_LIMIT_SWITCH Interruptor límite derecho del eje X. Valor = 0x2C.

Y_LEFT_AXIS_HARD_LIMIT_SWITCH Interruptor límite izquierdo del eje Y. Valor = 0x2D.

Y_RIGHT_AXIS_HARD_LIMIT_SWITCH Interruptor límite derecho del eje Y. Valor = 0x2E.

Z_LEFT_AXIS_HARD_LIMIT_SWITCH Interruptor límite izquierdo del eje Z. Valor = 0x2F.

Z_RIGHT_AXIS_HARD_LIMIT_SWITCH Interruptor límite derecha del eje Z. Valor = 0x30.

X_AXIS_CURRENT_POSITION Valor de la posición actual correspondiente al eje X. Valor = 0x20.

Y_AXIS_CURRENT_POSITION Valor de la posición actual correspondiente al eje Y. Valor = 0x21.

Z_AXIS_CURRENT_POSITION Valor de la posición actual correspondiente al eje Z. Valor = 0x22.

X_AXIS_MOTION_TRAVEL_RANGE Rango de desplazamiento correspondiente al eje X. Valor = 0x3D.

Y_AXIS_MOTION_TRAVEL_RANGE Rango de desplazamiento correspondiente al eje Y. Valor = 0x3E.

Z_AXIS_MOTION_TRAVEL_RANGE Rango de desplazamiento correspondiente al eje Z. Valor = 0x3F.

X_AXIS_REFERENCE_POINT_RETURN_DIRECTION Punto de referencia de la dirección de retorno del eje X. Valor = 0x40.

Y_AXIS_REFERENCE_POINT_RETURN_DIRECTION Punto de referencia de la dirección de retorno del eje Y. Valor = 0x41.

Z_AXIS_REFERENCE_POINT_RETURN_DIRECTION Punto de referencia de la dirección de retorno del eje Z. Valor = 0x42.

CUTTING_FEEDRATE_OVERRIDE Corrección de la velocidad de avance de corte. Valor = 0x1C.

RAPID_TRAVERSE_RATE Velocdad de avance. Valor = 0x38.

RAPID_TRAVERSE_RATE_OVERRIDE Corrección de a velocidad de avance. Valor = 0x1E.

MAXIMUM_TRAVERSE_RATE Velocidad máxima de avance. Valor = 0x3C.

JOG_FEEDRATE Velocidad de avance constante. Valor = 0x36.

JOG_FEEDRATE_OVERRIDE Corrección de la velocidad de avance constante. Valor = 0x1D.

MAXIMUM_JOG_FEEDRATE Velocidad máxima de avance constante. Valor = 0x37

FAST_ZRTR Velocidad rápida del avance de retorno a cero. Valor = 0x3A.

IV.28. Archivo javakiosk_.h

El archivo de encabezado *javakiosk_* contiene a la clase JavaKioskImpl encargada de la comunicación entre la interfaz gráfica de usuario implementada en Java y el resto de los módulos implementados en C++, es decir, es la interfaz entre el código Java y C++. Proporciona las interfaces TaskControl, Services y VirtualKeyboard, y extiende a la estructura Observer[22].

La información concerniente a los datos del sistema CNC se comunican en parejas de números enteros entre la interfaz gráfica de usuario en Java y el código en C++. El primer elemento corresponde al identificador del dato en cuestión y el segundo número es su respectivo valor. Los valores se encuentran definidos en los archivos milldatamodel.h y kbdcmdid.h, para el caso del código en C++. Por su parte, los archivos KbdCmdId.java y DataConst.java almacenan los mismos valores para el código en Java.

IV.28.1 Constructor

JavaKioskImpl JavaKioskImpl()

Crea un nuevo objeto de tipo JavaKioskImpl que representa la interfaz entre el código Java y C++. Establece el identificador y nombre de las interfaces proporcionadas e inicializa los campos.

IV.28.2 Métodos protegidos

getTaskControl cnc::base::TaskControl *getTaskControl()

Proporciona un apuntador a la interfaz TaskControl proporcionada por la clase.

getServices Services *getServices()

Proporciona un apuntador a la interfaz Services proporcionada por la clase.

getVirtualKeyboard VirtualKeyboard *getVirtualKeyboard()

Proporciona un apuntador a la interfaz VirtualKeyboard proporcionada por la clase.

[22] Se trata del patrón observador y se explica en el apéndice correspondiente.

setPersistentData void setPersistentData(cnc::base::MillPersistentData *pd)
Establece el apuntador que permite el acceso a los datos persistentes del sistema.

Parametros:

`pd` - apuntador a un ejemplar de la clase MillPersistentData encargada de los datos persistentes del sistema.

setGlobalData void setGlobalData(cnc::base::GlobalData *gd)
Establece el apuntador que permite el acceso a los datos de la memoria del sistema. Además, agrega observadores que notifican de actualizaciones en los datos.

Parámetros:

`gd` - apuntador a un ejemplar de la clase GlobalData encargada de los datos de la memoria del sistema.

setProgramManager void setProgramManager(cnc::compiler::ProgramManager *pm)
Establece el apuntador que permite el acceso a métodos encargados de las tareas de edición y administración de archivos de programas.

Parámetros:

`pm` - apuntador a un ejemplar de la estructura ProgramManager encargada de las tareas de edición y administración de archivos de programas.

setEnvironmentEntities void setEnvironmentEntities(cnc::rtl::EnvironmentEntities *env)
Establece el apuntador al ejemplar de la clase EnvironmentEntities.

update virtual void update(const char *registerID)
Recibe y comunica las actualizaciones de los datos que cuenten con observadores, establecidos en el método setGlobalData, a la interfaz gráfica de usuario.

Parámetros:

`registerID` - apuntador a una cadena de caracteres contenedora del identificador de un dato actualizado.

start bool start()
Implementación del método start propuesto por la interfaz TaskControl. Inicializa la maquina virtual de Java, crea un ejemplar de la clase StartUIManager encargada de establecer el el tema visual del entorno gráfico de Java y crea un ejemplar de la clase CoreGui que inicializa la GUI y es encargada de proporcionar métodos importantes para la interacción de la GUI con el código nativo. Inicializa los identificadores de estos métodos, necesarios para su invocación desde el código en C++, y también llama al método `init` encargado de comunicar a la GUI los valores iniciales de los datos del sistema. Retorna `true` si el inicio de la GUI es correcto y `false` en otro caso.

setInfoFiles void setInfoFiles()
Comunica a la GUI la información correspondiente a los archivos de programas la cual es necesaria para el modo de operación administrador de archivos. La información que se establece corresponde al nombre, tamaño y fecha de modificación del archivo.

init void init()

Comunica a la GUI los valores iniciales de los datos del sistema.

execute bool execute()

Implementación del método `execute` propuesto por la interfaz TaskControl. Se encuentra a la espera de cambios en el estado de la GUI, de los datos editables del sistema y tareas de edición y administración de archivos de programas. Retorna `false` si la GUI no se puede ejecutar correctamente y `true` en cualquier otro caso.

stop bool stop()

Implementación del método `stop` propuesto por la interfaz TaskControl. Se encuentra a la espera de la finalización de la interfaz gráfica liberando los recursos de la máquina virtual de Java. Retorna `false` si la GUI se sigue ejecutando y `true` en otro caso.

canExecute bool canExecute()

Proporciona el estado de la GUI, `true` si se ejecuta sin problemas y `false` en otro caso.

selectForm bool selectForm(const char *formID)

Establece el modo de operación correspondiente al identificador proporcionado en el parámetro.

Parámetros:

`formID` - identificador del modo de operación.

setInfo bool setInfo(const char *infoID, const char *utf8text)

setCommand void setCommand(const cnc::comm::KbdCommand kbdCmd)

Comunica comandos del teclado a la GUI.

Parámetros:

`kbdCmd` - comando del teclado que es enviado.

getCommand bool getCommand(cnc::comm::KbdCommand *kdbCmd)

Se encuentra a la espera de comandos generados por el teclado virtual. Retorna `true` si existe un comando a la espera y `false` en otro caso.

Parámetros:

`kdbCmd` - apuntador al comando recibido.

setParameterValue void setParameterValue(int id, int value)

Establece valores de los datos persistentes.

Parámetros:

`id` - valor del identificador del dato.

`value` - valor asignado.

setRegisterValue void setRegisterValue(int id, int value)
Establece valores de los datos de la memoria.

Parámetros:

`id` - valor del identificador del dato.

`value` - valor asignado.

setState bool setState(int category, int option)

getState int getState(int category, int option = -1)

IV.28.3 Enumeraciones

TASKS_FILES enum TASKS_FILES
Enumeración análoga a Task de la clase Mode en el código Java.

IV.28.4 Campos constantes

USER_CLASSPATH Ruta de acceso a las clases de la interfaz gráfica de usuario de Java.

IV.28.5 Campos protegidos

count_ unsigned count_
Campo encargado de informar el estado del módulo de interfaz gráfica. Si es igual a cero entonces el módulo no se ha iniciado correctamente, en otro caso significa que el inicio es correcto.

pd_ cnc::base::MillPersistentData *pd_
Apuntador a un ejemplar de la clase MillPersistentData encargada de los datos persistentes del sistema.

gd_ cnc::base::GlobalData *gd_
Apuntador a un ejemplar de la clase GlobalData encargada de los datos de la memoria del sistema.

pm_ cnc::compiler::ProgramManager *pm_
Apuntador a un ejemplar de la estructura ProgramManager encargada de las tareas de edición y administración de archivos de programas.

env_ cnc::rtl::EnvironmentEntities *env_
Apuntador a un ejemplar de la clase EnvironmentEntities.

jvm JavaVM *jvm
Apuntador a la máquina virtual de Java.

env JNIEnv *env
Apuntador al arreglo de apuntadores de funciones de JNI.

obj_core_gui jobject obj_core_gui
Referencia a un ejemplar de la clase CoreGui de la GUI Java.

class_core_gui jclass class_core_gui
Referencia a la clase CoreGui de la GUI Java.

id_changeExists jmethodID id_changeExists
Identificador del método changeExists de la clase CoreGui de la GUI Java.

id_newTasksExists jmethodID id_newTasksExists
Identificador del método newTasksExists de la clase CoreGui de la GUI Java.

id_getCommand jmethodID id_getCommand
Identificador del método getCommand de la clase CoreGui de la GUI Java.

id_setCommand jmethodID id_setCommand
Identificador del método setCommand de la clase CoreGui de la GUI Java.

id_selectForm jmethodID id_selectForm
Identificador del método selectForm de la clase CoreGui de la GUI Java.

id_existChangedData jmethodID id_existChangedData
Identificador del método existChangedData de la clase CoreGui de la GUI Java.

id_getDataValue jmethodID id_getDataValue
Identificador del método getDataValue de la clase CoreGui de la GUI Java.

IV.29. Resultados de las pruebas

La GUI puede ejecutarse por separado, sin la necesidad de tener el resto de los módulos, sin embargo, no hace gran cosa por si sola. Lo único que se obtendría al ejecutarla, serían dos ventanas, una completamente vacía y otra conteniendo solo el menú de modos de operación y el teclado virtual, sin alguna función como se muestra en la figura 4.44.Por otra parte, en la figura 4.45 se puede observar la GUI cuando inicia invocada desde el código en C junto con el resto de los módulos. Sobre la ventana superior se puede ver la consola de depuración, la cual permite visualizar la información que recibe el código en C al manipular la GUI. También se puede observar que el primer comando generado por el código en C fija el modo de retorno a cero.

En las capturas de la figura 4.47 se observan las salidas generadas al presionar los tres botones de retorno a cero de los ejes con las cadenas "*X axis homed*", "*Y axis homed*"y "*Z axis homed*". En general todas las entradas realizadas desde la GUI generan salidas semejantes por la consola de depuración. De esta forma se comprueba que la comunicación entre la GUI y el resto de los módulos funciona. Además, existe una comunicación recíproca entre los módulos y la GUI. Por ejemplo, en la figura 4.46, se puede observar como al manipular el control deslizante de la corrección de avance, la información es recibida por el código en C (como se ve en la consola) y el código en C vuelve a comunicar los cambios realizados en dicho parámetro a la GUI, la cual actualiza la barra que visualiza la corrección de avance con la información recibida. Al existir este tipo de comunicación, se pueden cambiar el teclado y los controles virtuales por unos físicos si así se desea y la interfaz sigue funcionando.

A continuación se muestran las tablas que contienen los resultados de las pruebas de caja negra planteadas en la metodología. Durante las pruebas se detectaron faltas, por lo tanto, estos resultados son los obtenidos después de realizar las correcciones pertinentes al código.

Los resultados de las pruebas se basan en lo visualizado en la consola de depuración de los módulos en C, no son resultados vistos directamente en una máquina CNC.

Tabla 4.15: Resultados de los casos de prueba de la categoría operación de la máquina

Nº de caso	Resultados
1	Se establece el modo memoria
2	Se establece el modo IMD
3	Se establece el modo administración de archivos
4	Se establece el modo editor.
5	Se establece el modo manivela.
6	Se establece el modo avance constante.

Tabla 4.16: Resultados de los casos de prueba de la categoría movimiento de los ejes

Nº de caso	Resultados
1	Se retorna al origen el eje x, se establecen la corrección de la velocidad de avance rápido en 0 %, la velocidad de avance constante en 0 mm/min y su corrección en 0 %, factor de manivela en 1 y se selecciona el eje x.
2	Se retorna al origen el eje y, se establecen la corrección de la velocidad de avance rápido en 120 %, la velocidad de avance constante en 200 mm/min y su corrección en 120 %, factor de manivela en 10 y se selecciona el eje y, por ultimo se establece la corrección de velocidad de avance de corte igual a 0 %.
3	Se retorna al origen el eje z, se establecen el factor de manivela en 100 y se selecciona el eje y, por ultimo se establece la corrección de la velocidad de avance de corte igual a 120 %.
4	Se mueven en dirección positiva los ejes *x*, *y* y *z*.
5	Se mueven en dirección negativa los ejes *x*, *y* y *z*.
6	Se establecen en cero la velocidad de avance de corte por omisión y la velocidad de avance contante, así como sus respectivos valores máximos y también la velocidad de avance rápido del retorno a cero.

Tabla 4.17: Resultados de los casos de prueba de la categoría movimiento de los ejes (Cont.)

Nº de caso	Resultados
7	Se establecen en 4294 mm los campos de la velocidad de avance de corte por omisión, de avance constante y avance rápido de retorno a cero, así como sus respectivos valores máximos.
8	Se establecen los rangos de desplazamiento de los ejes en cero y la dirección de retorno como negativa.
9	Se establecen los rangos de desplazamiento de los ejes en 2147 mm y la dirección de retorno como positiva.
10	En todos los campos editables, de la categoría movimiento de los ejes, se introduce la cadena de texto “number”, entonces al perder el foco, cada campo regresa a su valor valido anterior.
11	En todos los campos editables, de la categoría movimiento de los ejes, se introduce el valor -1 y al intentar confirmar el cambio, el programa retorna el foco y establece el valor valido anterior a cada campo.
12	En todos los campos editables, de la categoría movimiento de los ejes, se introduce el valor 4295 y al intentar confirmar el cambio, el programa retorna el foco y establece el valor valido anterior a cada campo.

Tabla 4.18: Resultados de los casos de prueba de la categoría husillo

Nº de caso	Resultados
1	Se establece la rotación del husillo en dirección al movimiento de las manecillas del reloj, se inicia el movimiento con una velocidad de 0 rpm y una corrección de 50 %.
2	Se establece la rotación del husillo en dirección contraria al movimiento de las manecillas del reloj, se detiene el movimiento y se cambia la velocidad a 3000 rpm con una corrección de 120 %.
3	Se establecen en 0 rpm y 0 %, los límites mínimo y máximo de la velocidad y corrección del husillo, respectivamente.
4	Se establecen en 65,535 rpm y 255 %, los límites mínimo y máximo de la velocidad y corrección del husillo, respectivamente.
5	Se introduce la cadena “number” en los cuatro campos editables y al perder el foco se revierte la acción colocando el valor anterior.
6	En los cuatro campos editables se introduce el valor -1 y al intentar confirmar el cambio, el programa retorna el foco y establece el valor anterior.
7	Se introducen los valores 65,536 rpm y 256, en campos de los límites mínimo y máximo de la velocidad y corrección del husillo, respectivamente. Al intentar confirmar los cambios el programa retorna el foco y establece el valor anterior.

Figura 4.44: GUI sin comunicación con el resto de los módulos

Figura 4.45: Captura del inicio de la interfaz gráfica

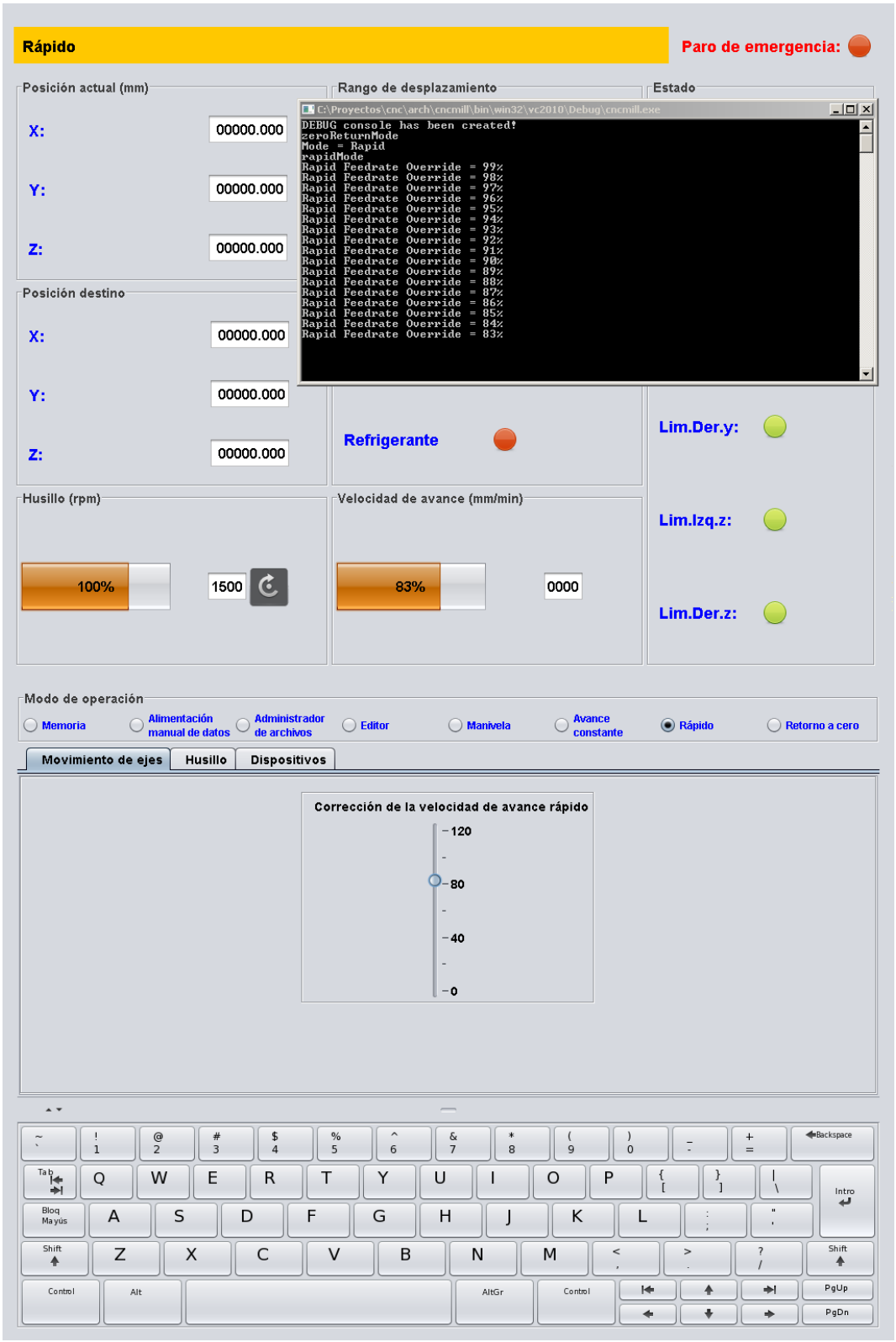
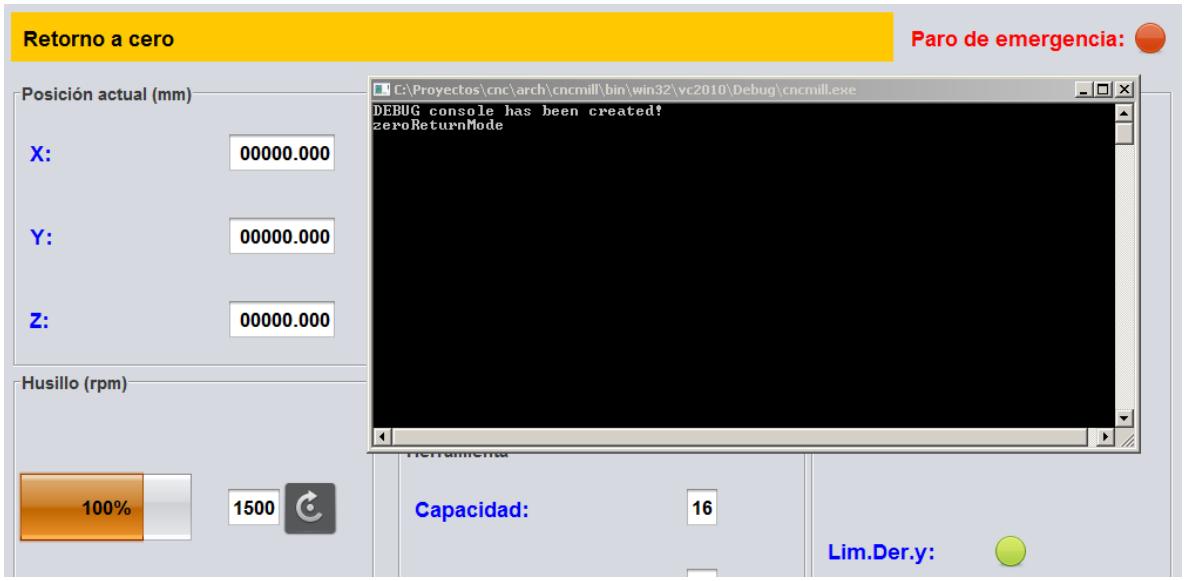
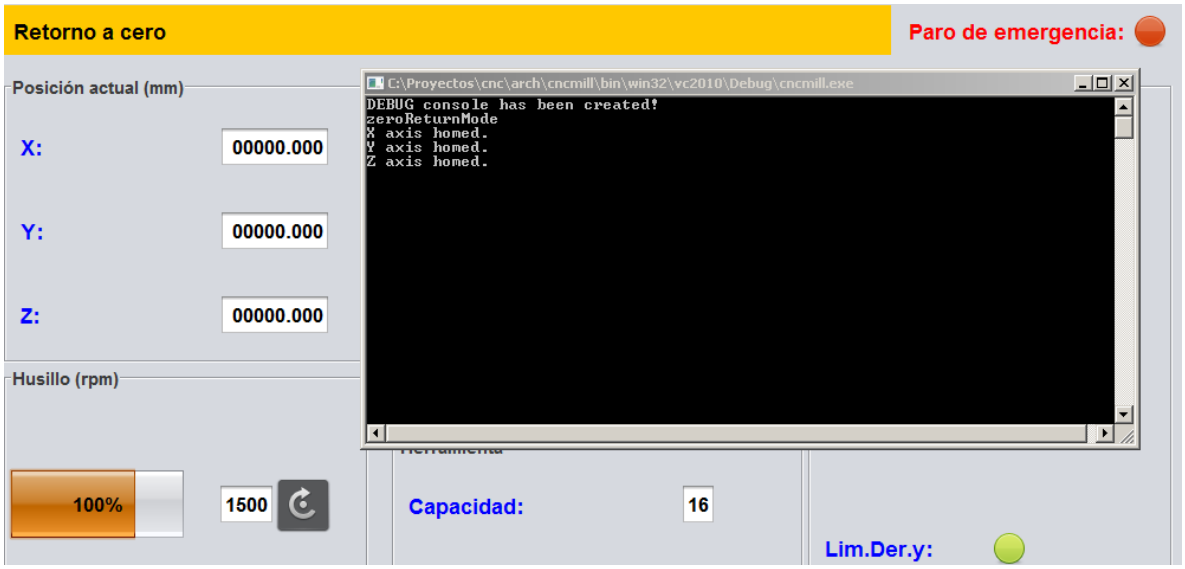


Figura 4.46: Comunicación recíproca entre los módulos



(a) Inicio de la aplicación.

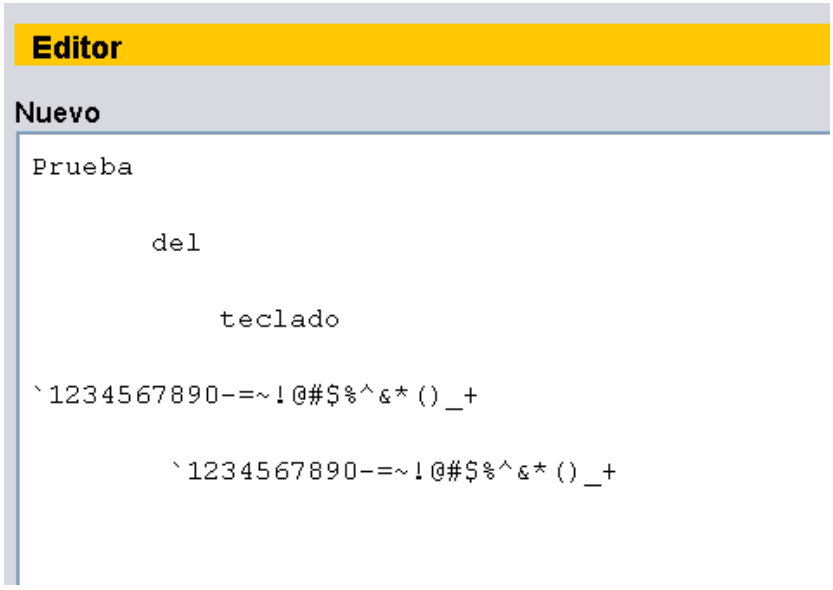


(b) Retorno a cero de los ejes.

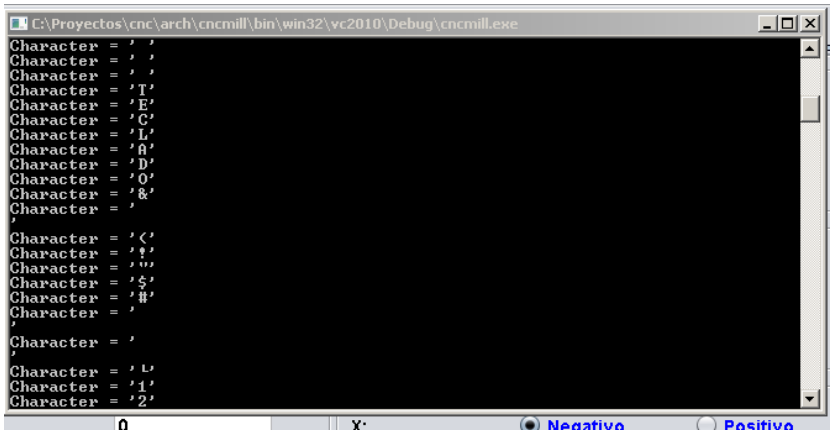
Figura 4.47: Salidas de la consola de depuración

Tabla 4.19: Resultados de los casos de prueba de la categoría dispositivos

Nº de caso	Resultados
1	Se desactivan los dispositivos.
2	Se activan los dispositivos.
3	La activación de los dispositivos se establece en automático.



(a) Captura del editor.



(b) Captura de la consola.

Figura 4.48: Capturas de las pruebas del modo editor

Tabla 4.20: Resultados de los casos de prueba de la categoría planificación de ruta

Nº de caso	Resultados
1	Se establece el sistema métrico internacional, el modo de dimensionamiento absoluto, el área de trabajo G54 con desplazamientos en los ejes de -2147 mm y a la herramienta activa se le establece una compensación de -2147 mm tanto en longitud como diámetro.
2	Se establece el sistema anglosajón, el modo de dimensionamiento incremental, el área de trabajo G60 con desplazamientos en los ejes de 2147 mm y a la herramienta activa se le establece una compensación de 2147 mm tanto en longitud como diámetro.
3	Se introduce la cadena “number” en los campos editables del desplazamiento del área de trabajo y compensación de la herramienta, y al perder el foco se revierte la acción colocando el valor anterior.
4	Se introduce el valor -2148 mm en los campos editables del desplazamiento del área de trabajo y compensación de la herramienta. Al intentar confirmar los cambios el programa retorna el foco y establece el valor anterior.
5	Se introduce el valor 2148 mm en los campos editables del desplazamiento del área de trabajo y compensación de la herramienta. Al intentar confirmar los cambios el programa retorna el foco y establece el valor anterior.

Tabla 4.21: Resultados de los casos de prueba de la categoría soporte de la herramienta

Nº de caso	Resultados
1	Se establece una dimensión de 0 mm en los campos de diámetro y longitud máximos de las herramientas.
2	Se establece una dimensión de 4294 mm en los campos de diámetro y longitud máximos de las herramientas.
3	Se introduce la cadena “number” en los campos editables de la categoría soporte de la herramienta, y al perder el foco se revierte la acción colocando el valor anterior.
4	Se introduce el valor -1 en los campos editables de la categoría soporte de la herramienta y al intentar confirmar los cambios el programa retorna el foco y establece el valor anterior.
5	Se introduce el valor 4295 en los campos editables de la categoría soporte de la herramienta y al intentar confirmar los cambios el programa retorna el foco y establece el valor anterior.

Tabla 4.22: Resultados de los casos de prueba de la categoría cambio automático de herramienta

Nº de caso	Resultados
1	Se establece la capacidad y el índice máximo del carrusel en cero, el tipo de selección como fijo y como herramienta activa ninguna.
2	Se establece la capacidad y el índice máximo del carrusel en 255, el tipo de selección como aleatorio y como herramienta activa la 255.
3	Se introduce la cadena “number” en los campos editables de la categoría cambio automático de herramienta, y al perder el foco se revierte la acción colocando el valor anterior.
4	Se introduce el valor -1 en los campos editables de la categoría cambio automático de herramienta y al intentar confirmar los cambios el programa retorna el foco y establece el valor anterior.
5	Se introduce el valor 256 en los campos editables de la categoría cambio automático de herramienta y al intentar confirmar los cambios el programa retorna el foco y establece el valor anterior.

Tabla 4.23: Resultados de los casos de prueba del teclado

Nº de caso	Resultados
1	Se teclea “ A ”, “ ‘ ”, arriba y borrar
2	Se teclea “ Z ”, “ - ”, abajo y tabulación
3	Se teclea “ 0 ”, “ = ”, izquierda y entrar
4	Se teclea “ 9 “, “ [“, derecha y bloq mayús
5	Se teclea “] “ y shift
6	Se teclea “ ; ” y control
7	Se teclea “ ’ “ y alt
8	Se teclea “ , “ y página arriba
9	Se teclea “ . “, “ / “ y página abajo

Tabla 4.24: Resultados de los casos de prueba de las tareas de edición y administración de archivos

Nº de caso	Resultados
1	Copia un archivo
2	Abre un archivo
3	Guarda un archivo
4	Guarda con un nombre distinto un archivo
5	Actualiza la información de los archivos en el directorio
6	Elimina un archivo
7	Cambia el nombre de un archivo

V. CONCLUSIONES

El desarrollo del presente trabajo muestra la implementación de una GUI para sistemas de CNC en el lenguaje Java la cual cumple con la condición de caja negra.

El protocolo de comunicación determinado por parejas de números enteros y el patrón *observer*, determina claramente las entradas y salidas del módulo. También se logra un comportamiento y sincronización correctos durante el intercambio de información.

La interfaz desarrollada en el archivo javakiosk_.h consigue conectar a la interfaz gráfica de usuario con el resto de los módulos en C++ haciendo uso del protocolo anterior junto con el API de invocación de Java.

Las pruebas de caja negra realizadas demuestran la buena integración con el resto de los módulos, puesto que las salidas mantienen consistencia en ralación a la información proporcionada.

Como se puede observar en el presente trabajo es posible realizar el desarrollo de un sistema de CNC en más de un lenguaje, en específico Java y C++, resultando en un sistema que se beneficia de las ventajas aportadas por cada uno de estos lenguajes.

V.1. Recomendaciones

Se buscó desarrollar el código de la interfaz gráfica de usuario lo mas sencillo posible y que fuera viable volver a usar parte del código para otros trabajos. Así, la adaptación de la presente interfaz gráfica de usuario en otros sistemas de CNC distintos a una fresadora debería resultar una tarea no muy complicada. Es necesario crear las categorías de datos que hagan falta o agregar nuevos datos a una categoría, junto con sus correspondientes componentes para visualizar y editar la información. También es necesario crear los modos de operación que hagan falta, con la configuración necesaria. La mayoría de estas tareas resulta en copiar y pegar código ya existente y cambiar las variables, así como agregar nuevos identificadores.

Se puede agregar un menú de opciones de estilo y tamaño de fuentes para adaptar mejor las pantallas a otras resoluciones de pantalla. Este menú funcionaría como editor del archivo de propiedades FontOptions. También es posible internacionalizar los diálogos de la GUI copiando y traduciendo el archivo StringsGui.

Para que la GUI sea aun mas intuitiva, se puede crear un juego de iconos que describa mejor cada uno de los controles, campos editables y visualizadores de los datos.

Por último, una ventaja de ser desarrollada la GUI en Java es su capacidad de incrustar aplicaciones en navegadores web como applets. Tal vez la presente interfaz pueda ser adaptada para ser usada en algún proyecto de teleoperación de sistemas de CNC.

PATRÓN OBSERVADOR

Es un patrón de diseño que define una dependencia uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y se actualicen automáticamente todos los objetos que dependen de él(Gamma et al. (2004)).

Un efecto secundario de dividir un sistema en una colección de clases cooperantes es la necesidad de mantener una consistencia entre objetos relacionados. Sin embargo se busca lograr esta consistencia sin que las clases terminen fuertemente acopladas para que sean reutilizables.

El patrón observador describe como establecer estas relaciones. Los principales objetos de este patrón son el **sujeto** y el **observador**. Un sujeto puede tener cualquier número de observadores dependientes de él. Cada vez el sujeto sufre cambios en su estado, todos los observadores son notificados. En respuesta, cada observador consultará al sujeto para sincronizar su estado con el estado de esté.

Este tipo de interacción es también conocida como publicar-suscribir. El sujeto es quien publica las notificaciones. Estas son enviadas sin saber quien las observa. Un número indeterminado de observadores puede suscribirse para recibir notificaciones.

Aplicación

El uso del patrón observador se puede dar en cualquiera de las siguientes situaciones:

- Cuando una abstracción tiene dos aspectos y uno dependiente del otro. Encapsular estos aspectos en objetos separados permite modificarlos y reutilizarlos de forma independiente.
- Cuando un cambio en un objeto requiere cambiar otros, y se desconoce el número total de objetos que necesitan cambiarse.
- Cuando un objeto debería ser capaz de notificar a otros sin hacer suposiciones sobre quiénes son dichos objetos. En otras palabras, cuando no queremos que estos objetos estén fuertemente acoplados.

Participantes

Sujeto
- Conoce a sus observadores. Un sujeto puede ser observado por cualquier número de objetos Observador.
- Proporciona una interfaz para asignar y quitar objetos Observador.

Observador
- Define una interfaz para actualizar los objetos que deben ser notificados ante cambios en un sujeto.

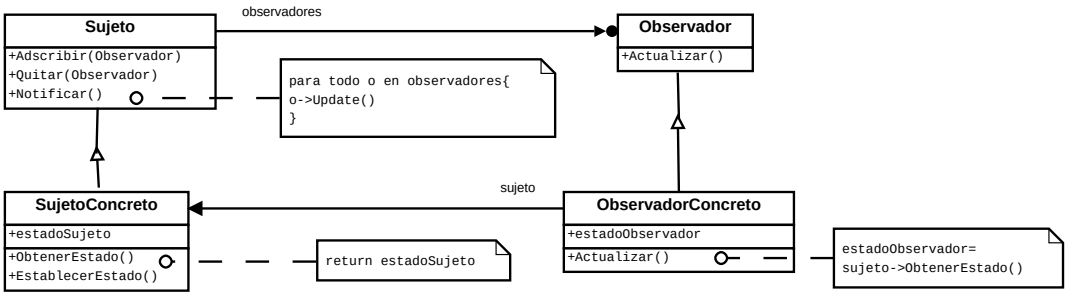


Figura 1: Estructura del patrón observador

SujetoConcreto
- Almacena el estado de interés para los objetos ObservadorConcreto.
- envía una notificación a sus observadores cuando cambia su estado.

ObservadorConcreto
- Mantiene una referencia a un objeto SujetoConcreto.
- Guarda un estado que debería ser consistente con el del objeto.
- Implementa la interfaz de actualización del Observador para mantener su estado consistente con el del sujeto.

GLOSARIO

A

API Application Programming Interface. Una interfaz de programación de aplicaciones es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

AWT Acrónimo en inglés de Abstract Window Toolkit. Se trata de un kit de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java.

C

cero pieza La posición que se hace de origen en el programa de pieza de una pieza de trabajo en particular. Esta posición es única en cada diseño de pieza de trabajo y es seleccionada por el programador de pieza.

CNC Acrónimo en inglés de Computer Numerical Control. Un control númerico por computadora es un tipo de sistema de control programable, dirigido por medio de datos matemáticos, el cual usa microcomputadores para realizar variadas operaciones de maquinado.

compensaciones Valores numéricos almacenados en los controles de CNC, el cual reposiciona los componentes de la máquina. Las compensaciones se usan para contrarrestar las variaciones en la geometría de la herramienta, el tamaño de la pieza, el desgaste de la herramienta, etc.

compensaciones de herramienta Un valor almacenado que compensa las variaciones en la longitud de la herramienta. Cada herramienta requiere de una compensación, la cual se mide a partir de la posición de la torreta o del husillo.

corrección Control que ajusta una velocidad o una proporción de avance programada. Esto mediante el uso de un porcentaje de dichas proporciones durante la operación.

código G Palabras formadas por la letra G seguida de un número que identifican funciones iniciales o tipos de ciclos en un sistema de control numérico.

D

desbastación Quitar las partes más bastas de una pieza que va a ser labrada.

DNC Acrónimo en inglés de Direct Numerical Control. El control numérico directo es un sistema que maneja máquinas de CNC por medio de un servidor central.

G

GUI Acrónimo en inglés de Graphical User Interface (Interfaz Gráfica de Usuario en español).

H

husillo La parte de la máquina herramienta que gira. En el centro de maquinado, el husillo sujeta una herramienta de corte. En el centro de torneado, el husillo sujeta la pieza de trabajo.

I

IU Una Interfaz de Usuario es un vínculo entre el usuario y un programa de computadora.

M

módulo Cada una de las partes de un programa que resuelve uno de los subproblemas en que se divide el problema complejo original.

P

paro de emergencia El control que apaga todas las funciones de la máquina automáticamente.

PMC Acrónimo en inglés de Programmable Motion Control (Control de Movimiento Programable).

V

velocidad de avance Velocidad en la que una herramienta de corte de torno se desplaza a lo largo de una pieza de trabajo.

W

widget Acrónimo de las palabras en inglés Window(ventana) y gaDGET(artefacto) y hace referencia a un objeto gráfico.

WIMP Acrónimo en inglés de Window, Icon, Menu, Pointer (ventana, icono, menú y puntero).

BIBLIOGRAFÍA

Equipo Editorial de Metalmecánica Internacional. Máquinas-herramienta en la industria metalmecánica mexicana, Octubre 2009. URL `http://www.metalmecanica.com/Articulos/`.

Industria y Comercio. Sistema de información arancelaria via internet, Mayo 2012. URL `http://www.economia-snci.gob.mx/siavi4/fraccion.php`.

Linux CNC. Linuxcnc.org, Mayo 2012. URL `http://www.linuxcnc.org/`.

Gilberto Herrera Ruiz and Arturo Molina. CHROM-1, desarrollo de un control numérico mexicano. *Ciencia y desarrollo*, 15(85):95–100, 1989.

P.J. Molina Moreno. *Especificación del interfaz de usuario: de los requisitos a la generación automática*. PhD thesis, Universidad Politécnica de Valencia, 2003.

C.M. Expósito. Interfaz gráfica de usuario: Aproximación semiótica y cognitiva. Technical report, Universidad de Laguna (España), 2006.

C. Rivera Loaiza. Utilización de redes de petri para la elaboración de una interfaz de usuario. Master's thesis, Universidad Michoacana de San Nicolás de Hidalgo, 2000.

ISO TC 184/SC 1. Numerical control of machines. operational command and data format. Technical report, ISO, 1981.

E. Villarreal. Prototipo cnc para el torneado en serie de metales. *Umbral Científico*, 1(12): 8–19, 2008.

S. Gordon and M.T. Hillery. Development of a high-speed cnc cutting machine using linear motors. *Journal of Materials Processing Technology*, 166(3):321–329, 2005.

J.A. Muriel Escobar and E. Giraldo Giraldo. Adecuación tecnológica de un torno compact 5 cnc a través de un pc. *Informador Técnico*, 74(74):7–13, 2010.

N.L. Ospina, P.L. Simanca, J.Á. Díaz, and E.M. Zapata. Descripción del diseno y construcción de un torno de control numérico. *Ingeniería y Ciencia*, 1(2):41–51, 2005.

H. Ji, Y. Li, and J. Wang. A software oriented cnc system based on linux/rtlinux. *The International Journal of Advanced Manufacturing Technology*, 39(3):291–301, 2008.

A.J. Álvarez and J.C.E. Ferreira. Webturning: Teleoperation of a cnc turning center through the internet. *Journal of materials processing technology*, 179(1):251–259, 2006.

C.S. Horstmann and G. Cornell. *Core Java 2 Volumen I - Fundamentos*, volume 1. Pearson Education, 7 edition, 2006a.

J.A. Luis and F.A. Matilde. *Java 2 Manual de Programación*. McGraw-Hill, España, 2001.

M. O'Connell. Java: The inside story. *SunWorld Online*, 7:Revista electrónica, 1995. Revista electrónica URL: http://www.sunworld.com/swol-07-1995/swol-07-java.html.

C.S. Horstmann and G. Cornell. *Core Java 2 Volumen II - Características avanzadas*, volume 2. Pearson Education, 7 edition, 2006b.

S. Liang. *The Java native interface: programmer's guide and specification*. Addison-Wesley Professional, 1999.

Oracle Corp. Java se 6 documentation, 2011. URL `http://docs.oracle.com/javase/6/docs/`.

Peter Smid. *CNC Programming Handbook: A Comprehensive Guide to Practical CNC Programming*. Industrial Press, Inc., New York, USA, 2 edition, 2003. ISBN 0-8311-3158-6.

Yoram Koren. *Computer Control of Manufacturing System*. McGraw-Hill, New York, USA, 1983.

Ian Sommerville. *Ingeniería del software*. Pearson Educación, 7 edition, 2005.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley, December 2004.