Universidad Nacional de Quilmes

TRABAJO FINAL DE CARRERA

Desarrollo e implementación de software traductor de archivos DXF a código G para fresadora EMCO PC Mill 55

Autores:
Celeste G. Guagliano,
Martín A. Retsinas

Director:
Ing. Diego Palmieri

Departamento de Ciencia y Tecnología

12 de julio de 2015

"Pensandolo bien, se que siempre supe el desenlace. Me pase la vida imaginandote, no es momento para ser cobarde..."

Gustavo Cerati - Nací para esto

UNIVERSIDAD NACIONAL DE QUILMES

Abstract

Departamento de Ciencia y Tecnología

Ingeniería en Automatización y Control Industrial

Desarrollo e implementación de software traductor de archivos DXF a código G para fresadora EMCO PC Mill 55

por Celeste G. Guagliano,
Martín A. Retsinas

Se presenta el diseño e implementación de DXF2Machine, un software CAM desarrollado en JAVA, multiplataforma y de código abierto, capaz de traducir archivos DXF a código máquina.
Se implementa DXF2Machine en la fresadora EMCO PC Mill 55 perteneciente a la Universidad Nacional de Quilmes. Para lograrlo se establece la comunicación, entre la PC que controla a la fresadora y la PC en donde corre DXF2Machine, mediante el montaje de una red LAN. Para la implementación de la red se instala una placa Ethernet en la PC que controla la fresadora. De ésta forma a través de la LAN se logra la transferencia de archivos de código traducido.
DXF2Machine se basa en la experiencia de mecanizado de piezas básicas en la industria metalmecánica, mediante el uso de distintos software de licencia propietaria. Dichos software son soluciones de elevado costo, esto ha llevado a idear DXF2Machine intentando presentar una alternativa accesible, de fácil uso y adaptable a cualquier tipo de máquina, tanto para aprendizaje como para PyMES que no pueden acceder a las opciones habituales.

Índice general

Índice de figuras

Índice de cuadros

A las personas que en cualquier parte arrancan virutas y mantienen la producción en marcha ...

Capítulo 1

Introducción

El desarrollo tecnológico ha llevado a la automatización de las máquinas herramienta convencionales, en las que todas las operaciones son accionadas mediante el uso de manivelas y palancas, dando lugar al surgimiento de las máquinas CNC (control numérico computarizado).
Las máquinas CNC son operadas mediante una computadora y apuntan a obtener mayor eficiencia en mecanizados.
Para optimizar tiempos, se ha hecho necesario automatizar la programación de dichas máquinas y de esta manera surgen los llamados software CAD-CAM.
A lo largo del presente proyecto se desarrolla un software CAM adaptado al tipo de programación de la fresadora EMCO PC Mill 55 perteneciente a la Universidad Nacional de Quilmes.
También se establece la comunicación entre el controlador de la fresadora y la computadora donde se encuentra el Software CAM instalado.
Por último se realizan diversas pruebas tanto en la fresadora EMCO PC Mill como en otras máquinas para demostrar la versatilidad del software desarrollado.

1.1. ¿Por qué realizar un traductor?

Las máquinas CNC tienen muchas ventajas sobre las máquinas convencionales. Las principales ventajas son: minimizar el error humano, repetibilidad, flexibilidad, reducir tiempos muertos, tiempos de mecanizado predecibles, etc.
Entre las principales desventajas del uso de máquinas por control numérico se encuentran: el elevado costo de la instalación y la programación de la misma.
La ejecución de las operaciones de mecanizado se realizan mediante el empleo de un conjunto de instrucciones (programa).Inicialmente el programa se realizaba “a pie de máquina” y esta tarea resulta muy engorrosa cuanto mas compleja es la geometría de la pieza. De aquí surge la necesidad de contar con un software, capaz de generar automáticamente este conjunto de instrucciones, a partir

de un plano de la pieza a mecanizar en formato digital (archivo). Dicho archivo es una representación a escala de la pieza en una determinada unidad de medición.

1.2. ¿Qué es un archivo DXF?

DXF, siglas en inglés de drawing exchange format, es un formato de archivo para dibujos de diseño asistido por computadora. Fue creado para posibilitar la interoperabilidad entre los archivos DWG, usados por el programa AutoCAD. Este tipo de archivos surgió en 1982, junto con la primera versión del programa AutoCAD, propiedad de Autodesk.
El formato DXF está basado en ASCII, o texto.

1.3. ¿Qué es y para que se utiliza el código G?

El código G (G-code) es el lenguaje de programación que habitualmente se utiliza en las máquinas herramienta CNC. Mediante este código la máquina recibe instrucciones sobre qué operación realizar y cómo debe llevarla a cabo.
Cada comando G Indica una condición de máquina o una acción a realizar. Dependiendo del tipo de control que tenga instalado la máquina (Fanuc, Fagor, Sinumerik, etc.) existen distintas variantes de código G.
Se obtiene un programa, conocido como programa NC, al agrupar un conjunto de bloques formados por una combinación de comandos G.
Algunos comandos G son universales independientemente del control utilizado, por ejemplo:

- G00: Movimiento rápido lineal
- G01: Interpolación lineal
- G02: Interpolación circular en sentido horario
- G03: Interpolación circular en sentido anti-horario ...

Un bloque de programa habitual es de la forma:

N02 G90 G01 X20 Y30 F500

En donde,
N: Número de bloque (opcional)

G90: Sistema de coordenadas absolutas
G01 X20 Y30: Coordenadas del punto final de la interpolación
F500: Velocidad de avance en milímetros por minuto

1.4. ¿En qué consiste un traductor?

El traductor es un software capaz de abrir un archivo de extensión DXF con la geometría a mecanizar y, luego de configurar diferentes parámetros, traducir este plano a código máquina.
A este tipo de software se lo conoce como CAM (fabricación asistida por computadora), por sus siglas en inglés

Capítulo 2

Objetivos

Como propósito principal del proyecto se plantea el desarrollo e implementación de un software capaz de generar un programa NC (código G), a partir de un archivo DXF, que pueda ser interpretado por la máquina EMCO PC MILL 55.
Como primera etapa se realiza el desarrollo y diseño del software traductor en un lenguaje de programación libre.
En forma simultánea se realiza la comunicación vía Ethernet entre la computadora que contiene dicho software y la computadora encargada de controlar la máquina. Mediante dicha comunicación se logra realizar la transferencia del programa NC.
Finalmente se procede a hacer una serie exhaustiva de pruebas para corroborar el correcto funcionamiento del software desarrollado y la comunicación establecida.

Capítulo 3

Estado del arte

3.1. Introducción al mecanizado, las fresadoras y las máquinas CNC

El mecanizado[1] es un proceso de fabricación que comprende un conjunto de operaciones de conformación de piezas mediante la eliminación de material, ya sea por arranque de viruta o por abrasión. Se realiza a partir de productos semielaborados como lingotes, tochos u otras piezas previamente conformadas por otros procesos como moldeo o forja. Los productos obtenidos pueden ser finales o semielaborados que requieran operaciones posteriores.
En particular el fresado es un tipo de mecanizado por arranque de viruta.

3.1.1. Mecanizado por arranque de viruta

El material es arrancado o cortado con una herramienta dando lugar a un desperdicio o viruta. La herramienta consta, generalmente, de uno o varios filos o cuchillas que separan la viruta de la pieza en cada pasada.
En el mecanizado por arranque de viruta se dan principalmente dos procesos:

- Desbaste: Eliminación de mucho material con poca precisión. Es un proceso intermedio.
- Acabado: Eliminación de poco material con mucha precisión. Es un proceso final cuyo objetivo es el de dar el acabado superficial que se requiera a las distintas zonas de la pieza.

3.1.2. Movimientos de corte

En el proceso de mecanizado por arranque de viruta intervienen dos movimientos:

- Movimiento principal: es el responsable de la eliminación del material.
- Movimiento de avance: es el responsable del arranque continuo del material, marcando la trayectoria que debe seguir la herramienta en tal fin.

Cada uno de estos movimientos puede ser realizado por la pieza o la herramienta según el tipo de mecanizado.

3.1.3. Mecanizado con máquina herramienta

El mecanizado se hace mediante una máquina herramienta[2], manual, semiautomática o automática, pero el esfuerzo de mecanizado es realizado por un equipo mecánico, con los motores y mecanismos necesarios. Las máquinas herramientas de mecanizado clásicas son: taladro, limadora, mortajadora, cepilladora, brochadora, torno, fresadora.
Desde hace ya tiempo, la informática aplicada a la automatización industrial, ha hecho que la máquina-herramienta evolucione hacia el control numérico. Así pues hablamos de centros de mecanizado de 5 ejes y tornos multifunción, que permiten obtener una pieza compleja, totalmente terminada, partiendo de un tocho o de una barra de metal y todo ello en un único amarre.
Estas máquinas con control numérico, ofrecen versatilidad, altas capacidades de producción y preparación, ofreciendo altísima precisión del orden de micras.
El presente trabajo se enfoca en las fresadoras CNC.

3.1.4. Fresadoras

Una fresadora[3] es una máquina herramienta utilizada para realizar trabajos mecanizados por arranque de viruta. En la figura 3.1 se muestra una fresadora universal.
El arranque de viruta se genera mediante el movimiento de una herramienta rotativa de varios filos de corte denominada fresa, figura 3.2.
Mediante el fresado es posible mecanizar los más diversos materiales, como madera, acero, fundición de hierro, metales no férricos y materiales sintéticos, superficies planas o curvas, de entalladura, de ranuras, de dentado, etc. Además las piezas fresadas pueden ser desbastadas o afinadas. En las fresadoras tradicionales, la pieza se desplaza acercando las zonas a mecanizar a la herramienta, permitiendo obtener formas diversas, desde superficies planas a otras más complejas.
Inventadas a principios del siglo XIX, las fresadoras se han convertido en máquinas básicas en el sector del mecanizado. Gracias a la incorporación del control numérico, son las máquinas-herramienta más polivalentes por la variedad de mecanizados que pueden realizar y la flexibilidad que permiten en el proceso de fabricación. La diversidad de procesos mecánicos y el aumento de la competitividad global han dado lugar a una amplia variedad de fresadoras que, aunque tienen una base común, se diferencian notablemente según el sector industrial en el que se utilicen. Asimismo, los progresos

FIGURA 3.1: Ejemplo de una Fresadora Universal

técnicos de diseño y calidad que se han realizado en las herramientas de fresar, han hecho posible el empleo de parámetros de corte muy altos, lo que conlleva una reducción drástica de los tiempos de mecanizado.

Debido a la variedad de mecanizados que se pueden realizar en las fresadoras actuales, al amplio número de máquinas diferentes entre sí, tanto en su potencia como en sus características técnicas, a la diversidad de accesorios utilizados y a la necesidad de cumplir especificaciones de calidad rigurosas, la utilización de fresadoras requiere de personal calificado profesionalmente, ya sea programador, preparador o fresador.

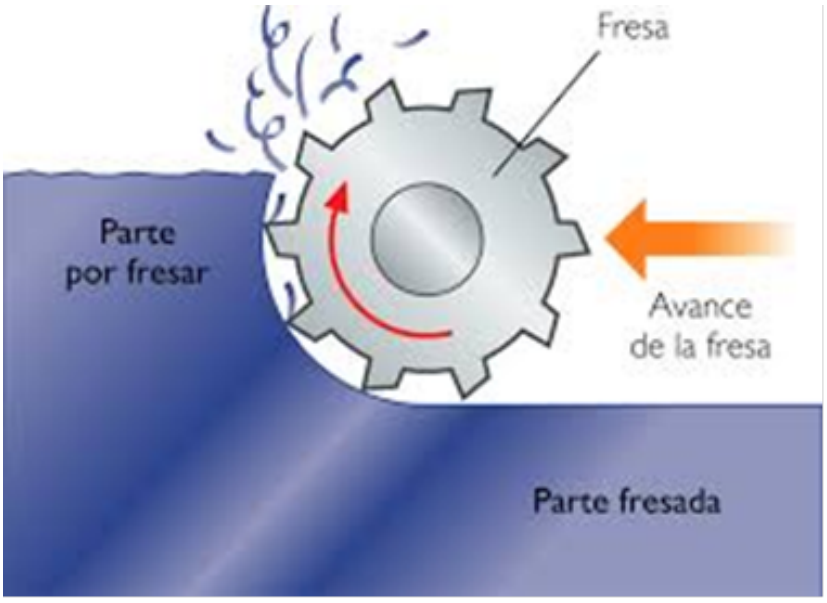


FIGURA 3.2: Herramienta de corte denominada Fresa

FIGURA 3.3: Ejemplo de Fresadora o Centro de Mecanizado CNC

3.1.5. Control numérico por computadora en fresadoras

Las fresadoras con control numérico por computadora (CNC) permiten la automatización programable de la producción. Se diseñaron para adaptar las variaciones en la configuración de productos. Su principal aplicación se centra en volúmenes de producción medios de piezas sencillas y en volúmenes de producción medios y bajos de piezas complejas, permitiendo realizar mecanizados de precisión con la facilidad que representa cambiar de un modelo de pieza a otro mediante la inserción del programa correspondiente y de las nuevas herramientas que se tengan que utilizar, así como el sistema de sujeción de las piezas. El equipo de control numérico se controla mediante un

programa que utiliza números, letras y otros símbolos; por ejemplo, los llamados códigos G (movimientos y ciclos fijos) y M (funciones auxiliares). Estos números, letras y símbolos, los cuales llegan a incluir &, %, $ y "(comillas), están codificados en un lenguaje apropiado para definir un programa de instrucciones para desarrollar una tarea concreta. Cuando la tarea en cuestión varía se cambia el programa de instrucciones. En las grandes producciones en serie, el control numérico resulta útil para la robotización de la alimentación y retirada de las piezas mecanizadas.
Las fresadoras universales modernas cuentan con visualizadores electrónicos donde se muestran las posiciones de las herramientas, según un sistema de coordenadas, y así se facilita mejor la lectura de cotas en sus desplazamientos. Asimismo, a muchas fresadoras se les incorpora un sistema de control numérico por computadora (CNC) que permite automatizar su trabajo. Además, las fresadoras copiadoras incorporan un mecanismo de copiado para diferentes perfiles de mecanizado.
Existen varios lenguajes de programación CNC para fresadoras, todos ellos de programación numérica, entre los que destacan el lenguaje normalizado internacional ISO y los lenguajes HEIDENHAIN, Fagor y Siemens. Para desarrollar un programa de CNC habitualmente se utilizan simuladores que, mediante la utilización de una computadora, permiten comprobar la secuencia de operaciones programadas. En la figura 3.3 se muestra un ejemplo de máquina CNC.

3.1.6. Operaciones de fresado

Con el uso creciente de las fresadoras de control numérico están aumentando las operaciones de fresado que se pueden realizar con este tipo de máquinas, siendo así que el fresado se ha convertido en un método polivalente de mecanizado. El desarrollo de las herramientas ha contribuido también a crear nuevas posibilidades de fresado además de incrementar de forma considerable la productividad, la calidad y exactitud de las operaciones realizadas.
El fresado consiste principalmente en el corte del material que se mecaniza con una herramienta rotativa de varios filos, que se llaman dientes, labios o plaquitas de metal duro, que ejecuta movimientos de avance programados, de la mesa de trabajo en casi cualquier dirección de los tres ejes posibles, en los que se puede desplazar la mesa donde va fijada la pieza que se mecaniza.
Las herramientas de fresar se caracterizan por su diámetro exterior, el número de dientes, el paso de los dientes (distancia entre dos dientes consecutivos) y el sistema de fijación de la fresa en la máquina. En las fresadoras universales utilizando los accesorios adecuados o en las fresadoras de control numérico se puede realizar la siguiente relación de fresados:

- Planeado: La aplicación más frecuente de fresado es el planeado, que tiene por objetivo conseguir superficies planas. Para el planeado se utilizan generalmente fresas de planear de plaquitas intercambiables de metal duro, existiendo una gama muy variada de diámetros de estas fresas y del número de plaquitas que monta cada fresa. Los fabricantes de plaquitas recomiendan como primera opción el uso de plaquitas redondas o con ángulos de 45º como alternativa. En la figura 3.4 se muestra un ejemplo de la realización de esta operación.

FIGURA 3.4: Realización de un planeado

- Cubicaje: La operación de cubicaje es muy común en fresadoras verticales u horizontales y consiste en preparar los tarugos de metal u otro material como mármol o granito en las dimensiones cúbicas adecuadas para operaciones posteriores. Este fresado también se realiza con fresas de planear de plaquitas intercambiables.

- Contorneado: En las fresadoras de control numérico además del cubicaje pueden realizarse operaciones mas complejas de contorneado, esta operación permite mecanizar el contorno exterior de una pieza determinada con mayor libertad en la geometría.

- Grabado: Esta operación en las fresadoras de control numérico consiste en el mecanizado en bajo relieve de las geometrías programadas.

- Taladrado, escariado y mandrinado: Estas operaciones se realizan habitualmente en las fresadoras de control numérico dotadas de un almacén de herramientas y utilizando las herramientas adecuadas para cada caso.

- Fresado en escuadra: El fresado en escuadra es una variante del planeado que consiste en dejar escalones perpendiculares en la pieza que se mecaniza. Para ello se utilizan plaquitas cuadradas o rómbicas situadas en el portaherramientas de forma adecuada.

- Corte: Una de las operaciones iniciales de mecanizado que hay que realizar consiste muchas veces en cortar las piezas a la longitud determinada partiendo de barras y perfiles comerciales de una longitud mayor. Para el corte industrial de piezas se utilizan indistintamente sierras de cinta o fresadoras equipadas con fresas cilíndricas de corte. Lo significativo de las fresas de corte es que pueden ser de acero rápido o de metal duro. Se caracterizan por ser muy delgadas (del orden de 3 mm aunque puede variar), tener un diámetro grande y un dentado muy fino. Se utilizan fresas de disco de relativamente poco espesor (de 0,5 a 6 mm) y hasta 300 mm de

diámetro con las superficies laterales retranqueadas para evitar el rozamiento de estas con la pieza.

- Ranurado recto: Para el fresado de ranuras rectas se utilizan generalmente fresas cilíndricas con la anchura de la ranura y, a menudo, se montan varias fresas en el eje portafresas permitiendo aumentar la productividad de mecanizado. Al montaje de varias fresas cilíndricas se le denomina tren de fresas o fresas compuestas. Las fresas cilíndricas se caracterizan por tener tres aristas de corte: la frontal y las dos laterales. En la mayoría de aplicaciones se utilizan fresas de acero rápido ya que las de metal duro son muy caras y por lo tanto solo se emplean en producciones muy grandes.

- Ranurado de forma: Se utilizan fresas de la forma adecuada a la ranura, que puede ser en forma de T, de cola de milano, etc.

- Ranurado de chaveteros: Se utilizan fresas cilíndricas con mango, conocidas en el argot como bailarinas, con las que se puede avanzar el corte tanto en dirección perpendicular a su eje como paralela a este.

- Copiado: Para el fresado en copiado se utilizan fresas con plaquitas de perfil redondo a fin de poder realizar operaciones de mecanizado en orografías y perfiles de caras cambiantes. Existen dos tipos de fresas de copiar: las de perfil de media bola y las de canto redondo o tóricas.

- Fresado de cavidades: En este tipo de operaciones es recomendable realizar un taladrado previo y a partir del mismo y con fresas adecuadas abordar el mecanizado de la cavidad teniendo en cuenta que los radios de la cavidad deben ser al menos un 15 % superior al radio de la fresa.

- Torno-fresado: Este tipo de mecanizado utiliza la interpolación circular en fresadoras de control numérico y sirve tanto para el torneado de agujeros de precisión como para el torneado exterior. El proceso combina la rotación de la pieza y de la herramienta de fresar siendo posible conseguir una superficie de revolución. Esta superficie puede ser concéntrica respecto a la línea central de rotación de la pieza. Si se desplaza la fresa hacia arriba o hacia abajo coordinadamente con el giro de la pieza pueden obtenerse geometrías excéntricas, como el de una leva, o incluso el de un árbol de levas o un cigüeñal. Con el desplazamiento axial es posible alcanzar la longitud requerida.

- Fresado de roscas: El fresado de roscas requiere una fresadora capaz de realizar interpolación helicoidal simultánea en dos grados de libertad: la rotación de la pieza respecto al eje de la hélice de la rosca y la traslación de la pieza en la dirección de dicho eje. El perfil de los filos de corte de la fresa deben ser adecuados al tipo de rosca que se mecanice.

- Fresado frontal: Consiste en el fresado que se realiza con fresas helicoidales cilíndricas que atacan frontalmente la operación de fresado. En las fresadoras de control numérico se utilizan

cada vez más fresas de metal duro totalmente integrales que permiten trabajar a velocidades muy altas.

- Fresado de engranajes: El fresado de engranajes apenas se realiza ya en fresadoras universales mediante el plato divisor, se realiza en máquinas especiales llamadas talladoras de engranajes y con el uso de fresas especiales del módulo de diente adecuado.
- Mortajado: Consiste en mecanizar chaveteros en los agujeros, para lo cual se utilizan brochadoras o bien un accesorio especial que se acopla al cabezal de las fresadoras universales y transforma el movimiento de rotación en un movimiento vertical alternativo.
- Fresado en rampa: Es un tipo de fresado habitual en el mecanizado de moldes que se realiza con fresadoras copiadoras o con fresadoras de control numérico.

En el presente informe nos centramos en los primeros 5 items.

3.1.7. Equipamiento de una fresadora de control numérico

Los equipamientos de serie y opcionales que montan las fresadoras actuales son muy variables en función de las prestaciones que tengan [1].
Respecto al manejo de la información, es necesario tener en cuenta el tipo de lenguaje de programación que es posible utilizar, la capacidad de memoria de la máquina para un uso posterior de los programas almacenados, así como la forma de introducción y modificación de los programas: a pie de máquina, mediante dispositivos de almacenamiento de datos (disquete o memoria USB), o mediante una tarjeta de red.
La unidad central de proceso (CPU, por sus siglas en inglés) de la máquina controla accionamientos rotativos, para lo cual se utilizan servomotores que pueden variar su velocidad en un rango continuo. El movimiento lineal de los carros de la mesa se obtiene transformando el movimiento rotacional de los servomotores mediante husillos de bolas sin juego.
La CPU obtiene datos del programa y de los sensores instalados, los cuales permiten establecer una realimentación del control de las operaciones. La precisión de estos sensores y la velocidad de procesamiento de la CPU limitan la precisión dimensional que puede obtenerse. El tipo de sensor utilizado ha evolucionado con el tiempo, siendo en la actualidad muy utilizados los sensores de efecto Hall para el control de los desplazamientos y giros realizados. Para controlar la posición del origen del sistema de referencia de los movimientos realizados y el desgaste de la herramienta se utilizan uno o varios palpadores o sondas de medida. Un palpador es un dispositivo con un vástago que acciona un pulsador al hacer contacto con la pieza o con la mesa de la máquina. También puede establecerse el origen de coordenadas realizando un contacto en movimiento de la herramienta con la zona a mecanizar.

Además de los movimientos de la pieza y de la herramienta, pueden controlarse de manera automatizada otros parámetros como la herramienta empleada, que puede cambiarse desde un almacén de herramientas instalado en la máquina; el uso o no de fluido refrigerante o la apertura y cierre de las puertas de seguridad.

3.2. Software CAD-CAM

CAD-CAM[4] son las siglas en inglés de:
Computer-Aided Design (CAD): Diseño asistido por computadora.
Computer-Aided Manufacturing(CAM): Fabricación asistida por computadora.
Los distintos software CAD permiten llevar a cabo de forma sencilla el diseño de las piezas a fabricar, y se complementan con los software CAM que permiten obtener de manera simplificada el o los programas NC necesarios para la fabricación de la pieza.
Existen actualmente en el mercado software que integran ambos módulos logrando así con la misma plataforma resolver ambas tareas. Dependiendo del tipo de rasgos que posibilitan mecanizar, los software CAD-CAM pueden clasificarse en:

- 2,5 ejes: Es el más básico de los Software CAM, permite generar el mecanizado de superficies planares.
- 3 ejes: Este tipo de software permite obtener piezas de mayor complejidad permitiendo que los 3 ejes se muevan con total libertad.
- 4 o más ejes: Se agregan grados de libertad en cuanto a inclinaciones o giros de los materiales a mecanizar, permitiendo entre otras cosas el torno-fresado.

3.2.1. Descripción de los software CAD-CAM más utilizados

En la actualidad los software de CAD-CAM más utilizados en la industria son:

- PowerMill[5]:Es un software CAM fabricado por DELCAM que corre bajo Microsoft Windows para la programación de fresas CNC desde 2 hasta 5 ejes. Se distribuye bajo licencia propietaria. Costo aproximado de la adquisición de la licencia U$S18000.-
- NX[6]: Formalmente conocido como NX Unigraphics o solo U-G, es un potente software CAD/CAM originalmente desarrollado por Unigraphics y desde 2007 por Siemens PLM Software. Es multiplataforma, corre bajo: Windows XP o posterior, Mac OS y sistemas UNIX.
  Se distribuye bajo licencia propietaria.Costo aproximado de la adquisición de una licencia U$S21500.-

- MasterCAM[7] : Es uno de los primeros software CAD/CAM que se desarrollaron, comenzó originalmente como un sistema CAM 2D con herramientas CAD integradas.
  Originalmente fue desarrollado por CNC Software Inc en 1983.
  Corre bajo Windows y se distribuye bajo licencia propietaria. Costo aproximado de la adquisición de una licencia U$S18000.-

- Catia[8] : (computer-aided three dimensional interactive application) es un programa informático de diseño, fabricación e ingeniería asistida por computadora comercial realizado por Dassault Systèmes. Es un software CAD/CAM disponible para Microsoft Windows, Solaris,IRIX y HP-UX.
  Provee una arquitectura abierta para el desarrollo de aplicaciones o para personalizar el programa. Las interfaces de programación de aplicaciones, CAA2 (o CAAV5), se pueden programar en Visual Basic y C++.
  Se distribuye bajo licencia propietaria. Costo aproximado de la adquisición de licencia: U$S17000.-

- Pitagoras: Es un sistema CAM desarrollado en Brasil por MicroCAM para programación de fresados en 2,5 ejes. Posee herramientas CAD muy básicas y es capaz de importar archivos DXF e IGES de otros sistemas.
  Corre bajo Windows y se distribuye bajo licencia propietaria. Costo aproximado de adquisición de una licencia: U$S2200.-

- Bob CAD-CAM[9]: Sistema CAM desarrollado para sistemas con Windows XP o posterior. Es compatible con diversos software CAD e incluso viene integrado en algunas versiones.
  Se distribuye bajo licencia propietaria. Costo aproximado de adquisición de una licencia U$S7000.-

- CAMWorks[10]: Software desarrollado por Geometric, viene integrado con SolidWorks entre otros sistemas de modelado 3D.
  Posee módulos para generar mecanizados de 2,5 hasta 5 ejes.
  Corre bajo Windows y se distribuye bajo licencia propietaria. Costo aproximado de adquisición de una licencia U$S 19000.-

- FreeMill: Diseñado por MECSOFT FreeMill es un software CAM que puede complementarse con un software CAD gratuito, como su nombre lo indica es de distribución libre y gratuita pero de igual forma posee licencia propietaria. Corre bajo Windows. Costo aproximado de adquisición de una licencia U$S0.-

- HeeksCNC[11]: Desarrollado principalmente para Linux, HeeksCNC es un módulo CAM diseñado para integrarse con HeeksCAD. Es un software de código abierto desarrollado en C++ y

python que posee además una versión para windows de distribución paga. Su instalación en Linux es muy engorrosa y no es fácilmente transportable de un sistema a otro. Valor aproximado de adquisición de una licencia en Windows U$S12.-

Capítulo 4

Investigación

4.1. EMCO PC Mill 55

4.1.1. Comunicación entre el controlador y la fresadora

La comunicación[12] entre el controlador y la fresadora se realiza por medio de una tarjeta de interface que va instalada en la PC de trabajo. En dicha PC se encuentra instalado el software WinNC que sirve de interacción entre el usuario y la fresadora. La instalación de la tarjeta de interface PCCOM debe realizarse antes de la instalación del software WinNC en el PC. La misma se coloca en un slot ISA que se encuentre disponible. Se debe conectar el cable de interface por el extremo "PC-X1" en el enchufe inferior de la tarjeta de interface, el cable NO debe ser conectado de manera inversa, y el otro extremo "Maschine-X300" en el armario eléctrico de la máquina.

4.1.2. PC que integra el sistema

La fresadora se encuentra conectada a una PC en donde corre el controlador de la misma.
A continuación se detallan las características mínimas y deseadas que requiere el sistema. Cabe aclarar que la tecnología involucrada en la informática esta en constante evolución y los equipos que se detallan en las características deseadas de la PC en la actualidad son difíciles de conseguir ya que muchos elementos fueron reemplazados por otros de mejores prestaciones. Es por eso que se presentan problemas de compatibilidad y el gran desafío de solventar estos problemas para mantener los equipos en funcionamiento.

Característica	Configuración Mínima	Recomendada
IBM o IBM-Compatible	PC Pentium 75	Pentium 100
Disco Duro	20 MB libres	
Unidad de Disco	Disquetera 3 1/2”	
Sistema Operativo	Windows 95	
Memoria RAM	8MB	16 MB
Tarjeta gráfica	VGA Color	
Pantalla	Color Resolución: 640 x 480	
Teclado	MF-2	
BUS (slots libres)	ISA o EISA BUS para la tarjeta de interface PCCOM (RS422/CAN)	

Cuadro 4.1: Requerimientos del sistema

4.1.3. Limitaciones

Se encuentran dos grandes limitaciones en el sistema:

- WinNC corre en las versiones de Windows 3.1 hasta Windows Millenium. Por lo tanto la PC que integra el sistema posee Windows 95 lo cual trae limitaciones de compatibilidad con sistemas actuales.
- Comunicación: La pc posee como único medio de comunicación con el mundo exterior una disquetera 3 1/2”

4.1.4. Software WinNC

El software EMCO WinNC es parte del concepto de entrenamiento de EMCO. En el mismo software pueden utilizarse distintos controladores para la fresadora. Esto hace posible conocer distintos métodos de programación y conocer las características particulares de cada controlador, ya que si bien todos cumplen la misma función, cada controlador tiene comandos particulares e incluso utilizan distintos parámetros para configurar el mismo mecanizado.
Actualmente la fresadora EMCO PC Mill 55 perteneciente a la Universidad Nacional de Quilmes cuenta con 3 opciones de controladores diferentes: Sinumerik 810, Sinumerik 820 y Fanuc. En la figura 4.1 se muestra la pantalla de inicio del software WinNC donde se debe seleccionar el control a utilizar.
A continuación se describe el controlador Sinumerik 810, ya que el teclado instalado en el control corresponde a este modelo.
Luego se describe el controlador HAAS/Fanuc que utiliza el centro de mecanizado HAAS VF3YT perteneciente a EMT y que se utiliza para pruebas del presente proyecto.

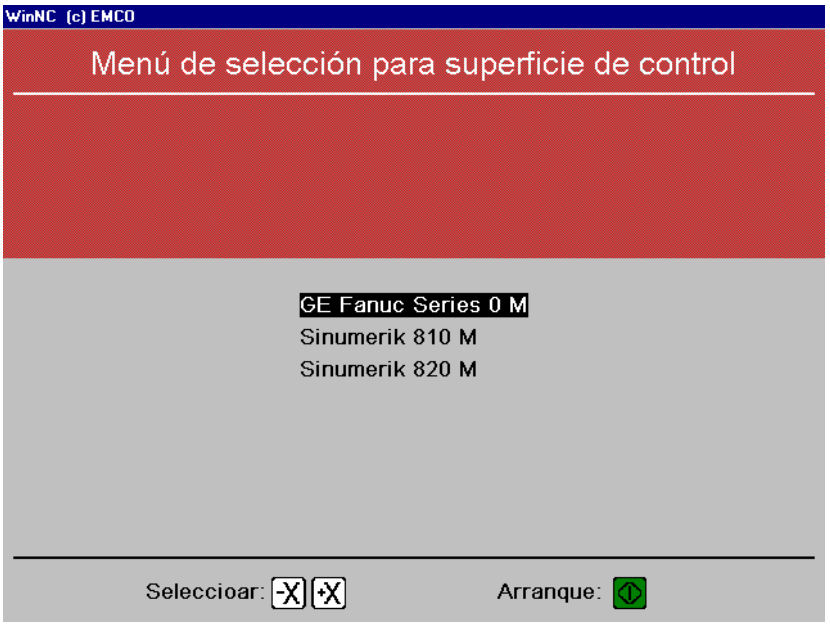


FIGURA 4.1: Selección del controlador a utilizar en WinNC

4.1.5. Controlador Sinumerik

En la presente sección se explican algunos conceptos del control Sinumerik 810M[13] utilizado. En la figura 4.2 se muestra el panel de control que se utiliza para ingresar información a la máquina. Todo lo realizable por dicho panel también puede llevarse a cabo desde el teclado, como puede verse en la figura 4.3

4.1.5.1. Modos operativos

- Ref point Este modo se utiliza para la aproximación al punto de referencia.
  Al llegar al punto de referencia, la indicación de valor real se coloca sobre el valor de las coordenadas del punto de referencia. Con ello, el control conoce la posición de la herramienta en la zona de trabajo.
  La aproximación al punto de referencia ha de realizarse en las situaciones siguientes: al conectar la máquina, luego de un corte de corriente, o de la activación de las alarmas "Aproximar punto de referencia" o "Punto de referencia no alcanzado", o si se han producido colisiones, o si los carros se agarrotan por sobrecarga.
- MDI: El control ejecuta el bloque introducido y borra después el buffer de memoria para nuevas entradas
- Automatic: Para la ejecución de un programa de piezas, el control llama en este modo operativo uno tras otro a los bloques y los evalúa.
- Jog: Con las teclas de dirección puede desplazarse manualmente la herramienta.

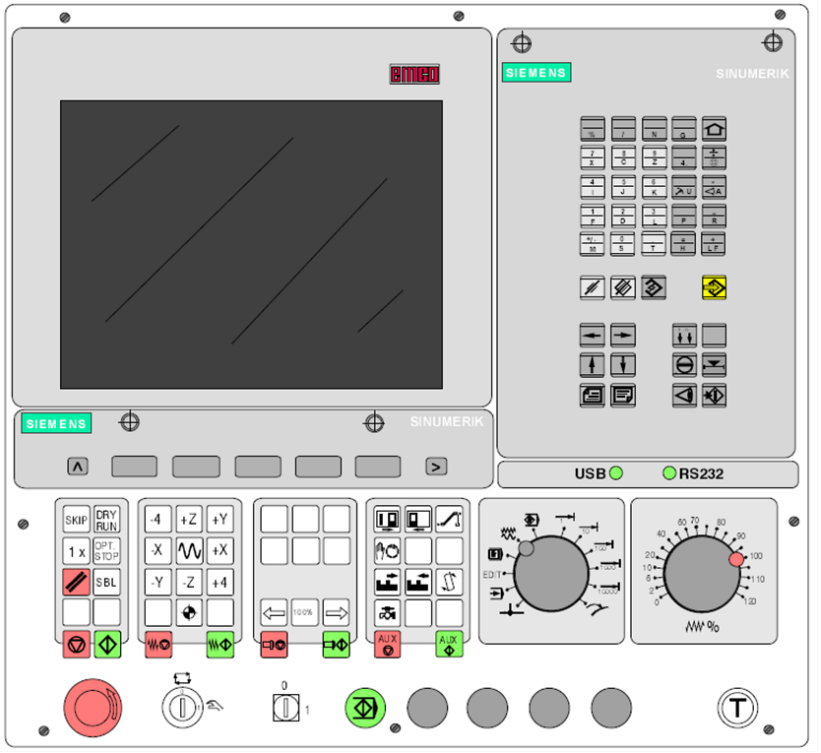


FIGURA 4.2: Panel de Control de Sinumerik 810M

FIGURA 4.3: Funcionamiento del teclado de la PC en Sinumerik 810M

4.1.5.2. Sistema de coordenadas

El sistema de coordenadas utilizado es universal (figura 4.4) y todos los controles lo utilizan en sus dos modalidades: absoluto e incremental.

4.1.5.3. Datos de herramientas

Cada herramienta (T) posee un número de identificación asociado.
Existe una base de datos (tabla) en la cual se cargan los datos de las herramientas. Cada control accede a dicha tabla, mediante el uso de distintas instrucciones, para tomar los datos de las herramientas. En la figura 4.5 se observan las principales características de las herramientas.

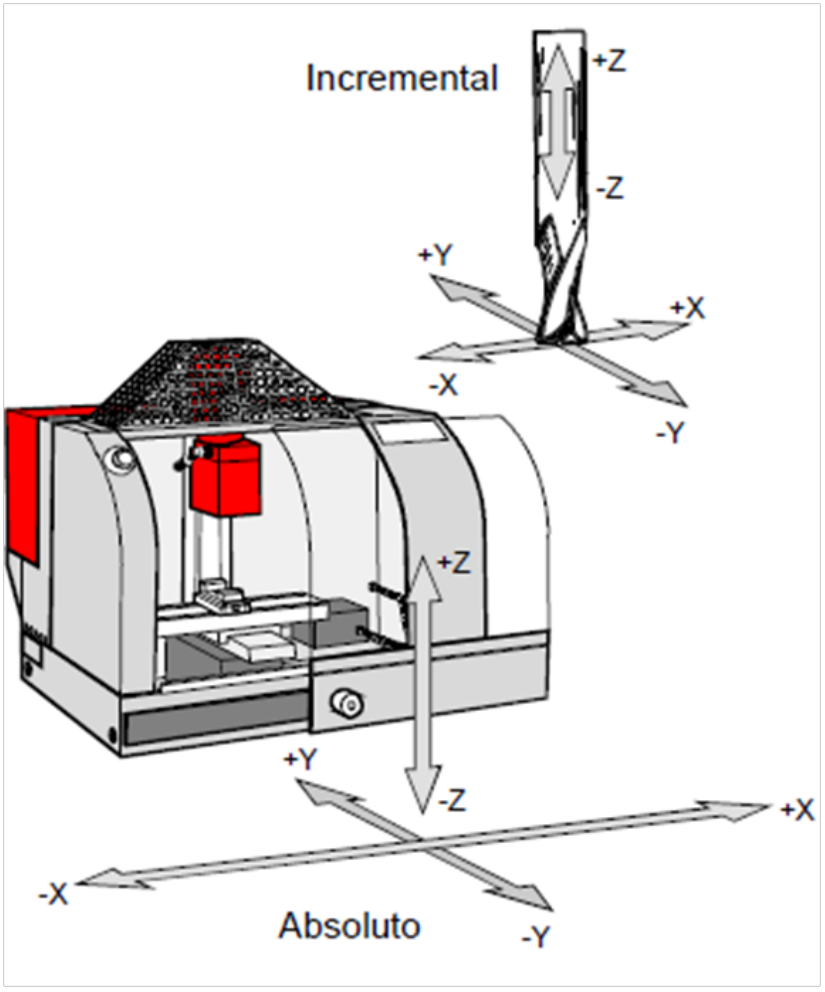


FIGURA 4.4: Diferencia entre Coordenadas Absolutas y Relativas

4.1.5.4. Decalaje de origen

El punto cero de coordenadas de máquina es desplazado al punto cero de pieza de trabajo, para facilitar la programación, mediante el empleo de funciones preparatorias (G54 – G57). Ver figura 4.6.

4.1.5.5. Programación

Se respetan los códigos G que se detallan bajo norma ISO-6983.
Cabe explicar el funcionamiento de los ciclos fijos de taladrado que son específicos de este controlador y se utilizan en la realización de la traducción:

- Ciclo fijo de taladrado: G81 R2=Zreferencia desde donde empieza el avance de taladrado R3=profundidad total del agujero R10=planoRetracción entre agujeros Favance.
  Ejemplo: G81 R2=5. R3=-25. R10=50. F70.

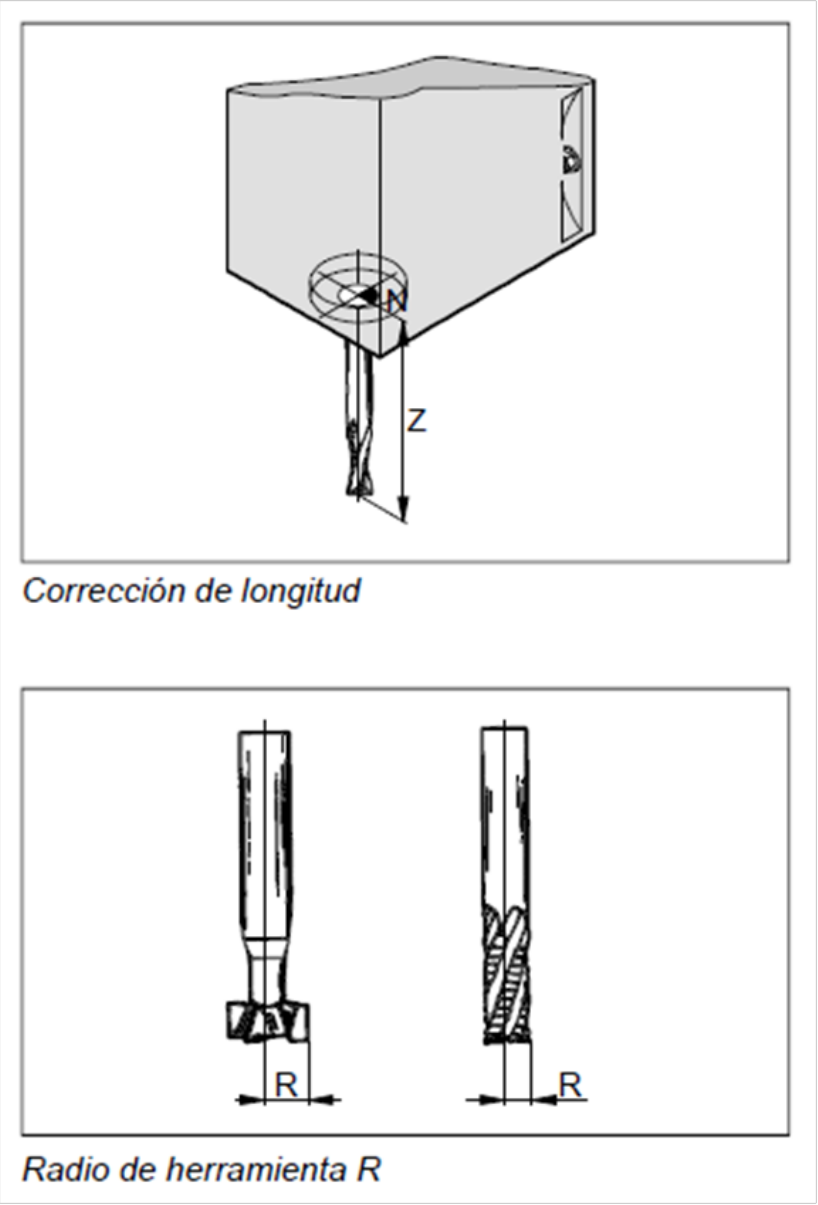


FIGURA 4.5: Datos utiles de las herramientas utilizadas

- Ciclo fijo de taladrado profundo: G83 R2=Zreferencia desde donde empieza el avance de taladrado R3=profundidad total del agujero R5=profundidad a incrementar por pasada R10=planoRetracción entre agujeros Favance.
  Ejemplo: G83 R2=5. R3=-25. R5=1.5 R10=50. F70.

4.1.6. Controlador Fanuc

El control Fanuc[14] es muy similar al Sinumerik, incluso en su método de programación. En la figura 4.7 se muestra el panel de control HAAS instalado en el centro de mecanizado VF3YT.
Una diferencia sustancial que se encuentra en la programación de HAAS/Fanuc con respecto a

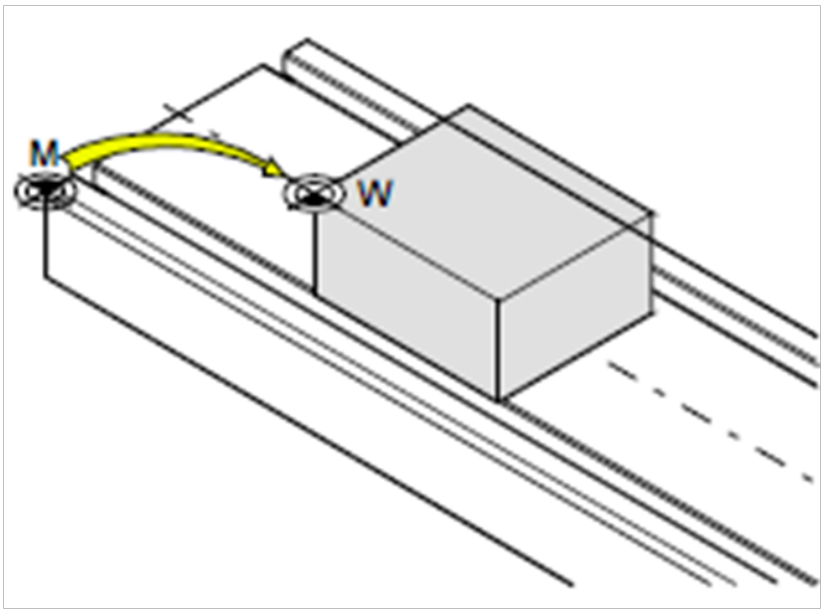


FIGURA 4.6: Cero de referencia en pieza o Decalaje de Origen

Sinumerik de EMCO es la forma de especificar un ciclo fijo de taladrado. A continuación se explican los ciclos fijos de taladrado que se utilizan en la traducción:

- Ciclo fijo de taladrado: G81 G98 (indica que la herramienta levanta hasta el punto de referencia entre agujeros) R(plano de referencia donde empieza el avance de taladrado) Z(profundidad del agujero) F(avance de corte).
  Ejemplo: G81 G98 R5. Z-25. F70.
- Ciclo fijo de taladrado profundo: G83 G98(indica hasta que plano debe levantar la herramienta entre agujeros) R(plano de referencia donde empieza el avance de taladrado) Z(profundidad del agujero) Q(pronfundidad a incrementar por pasada) F(avance de corte).
  Ejemplo: G83 G98 R5. Z-25. Q1.5 F70.

4.2. Comunicación

Considerando que la pc que integra el sistema solo cuenta con una disquetera 3 1/2çomo comunicación con el mundo exterior y visto que esta tecnología es muy vieja y difícil de utilizar con sistemas modernos debido a la dificultad de conseguir disquetes en buen estado. Se realiza un estudio sobre métodos alternativos para comunicar la PC que integra al sistema de la fresadora con la PC que hará de host para el software a diseñar.

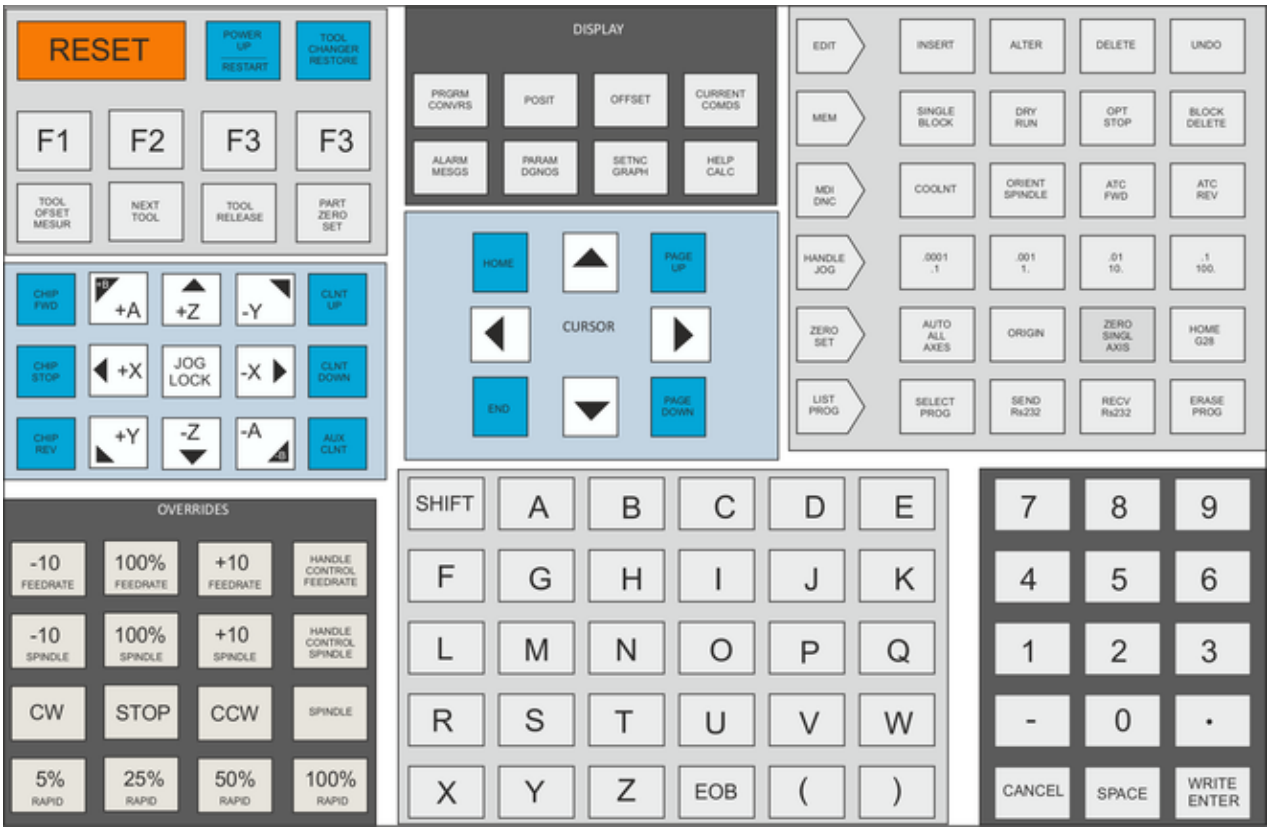


FIGURA 4.7: Panel de control en HAAS VF3YT

4.2.1. Ethernet vs USB

En la actualidad las unidades de almacenamiento externo han evolucionado y se reemplaza el uso de disquetes por memorias USB. Las cantidades de información y los tamaños de archivos manejados hoy en día obligaron a desarrollar unidades capaces de almacenar grandes cantidades de información en forma segura y rápida.
De la misma forma se cuenta con placas capaces de conectar dos o más PC entre sí, las denominadas placas Ethernet, para lograr transmitir gran cantidad de datos de forma rápida.
Considerando que la PC que integra el sistema utiliza disquetes y la PC en donde se aloja DXF2Machine no cuenta con esta tecnología, se evalúa la instalación de una placa USB o una placa Ethernet como alternativa a la disquetera a fin de resolver la comunicación.

4.2.1.1. Visión general del USB

"USB" significa "Universal Serial Bus", la versión 2.0 transmite datos a 480 megabits por segundo (Mb/s) o 60 megabytes por segundo (MB/s). Las velocidades de USB 2.0 son apropiadas para, por ejemplo, una grabadora de DVD externa o para un disco duro externo.
También se puede utilizar para transmitir potencia, que es como se enciende un concentrador USB. Una pequeña lámpara puede ser adaptada para extraer su energía de un puerto USB. Además, los

conectores USB se pueden conformar de seis maneras diferentes. Cada cable tiene un extremo “A” y un extremo “B”. Luego, está el conector estándar, el mini conector y el micro conector. Los dos últimos son muy populares para las cámaras digitales y otros dispositivos portátiles. Ethernet sólo utiliza un conector RJ-45 que es parecido a una toma de teléfono pero más grande.
Por desgracia en Windows 95 solo es posible utilizar un puerto USB para conectar periféricos, no permite el uso de unidades de almacenamiento externas. Recién a partir de Windows 98 se pueden usar memorias USB teniendo los drivers instalados de las mismas. Esto hace que se descarte la instalación de una placa USB.

4.2.1.2. Fundamentos de Ethernet

Ethernet no transmite energía, sólo datos. una conexión de gigabit Ethernet, es más rápida que USB 2.0. Gigabit Ethernet transfiere datos a velocidades de hasta 125 MB/s (un byte = ocho bits), lo que lo hace dos veces más rápido que el máximo teórico del USB 2.0.
Además, un cable Ethernet puede ir a unos 90 m sin degradar la calidad de la señal, mientras que un cable USB, por lo general, no puede pasar más de 3 m. Esa es una de las razones por las cuales Ethernet es el estándar para la conexión de dispositivos fijos de computación en un hogar o lugar de trabajo. Eso, y los cables de Ethernet son mucho más baratos que los de USB.
Los mothers más modernos cuentan con un controlador Gigabit Ethernet incluido, tal es el caso de la PC donde se encuentra DXF2Machine. Por lo que se opta por instalar una placa Ethernet ya que las mismas tienen compatibilidad con Windows 95 y hoy en día se siguen consiguiendo los drivers de instalación.

4.3. Almacenamiento de datos en archivos .DXF

En el articulo DXF Reference de Autodesk [15] se describe la forma de construcción de un archivo DXF. Para los alcances del presente proyecto se estudia en profundidad como se almacenan los datos de los “layers” especialmente del color y de 3 tipos particulares de entidades: circulo, linea y arcos:

4.3.1. layers

- Código 62: Número de color. Si se encuentra entre 0-255 define el color de la capa. Si el número es negativo significa que la capa está “apagada”, no puede visualizarse.

4.3.2. circulo

- Código 100: Marcador de subclase (AcDbCircle).
- Código 10: Valor de centro en X.
- Código 20,30: Valor de centro en Y,Z.
- Código 40: Radio.

4.3.3. línea

- Código 100: Marcador de subclase (AcDbLine).
- Código 10: Punto de inicio de línea en X.
- Código 20,30: Punto de inicio de línea en Y,Z.
- Código 11: Punto de finalización de línea en X.
- Código 21,31: Punto de finalización de línea en Y,Z.

4.3.4. arco

- Código 100: Marcador de subclase (AcDbCircle).
- Código 10: Valor de centro en X.
- Código 20,30: Valor de centro en Y,Z.
- Código 40: Radio.
- Código 100: Marcador de subclase (AcDbArc).
- Código 50: Ángulo de inicio.
- Código 51: Ángulo de finalización.

Por convención los arcos se toman en sentido antihorario, de esta forma resulta independiente la forma en que se guardan los datos de la forma en la que se dibujaron. Este es un dato no menor para la realización del software traductor. Los archivos DXF poseen muchas mas entidades y características particulares que los componen como ser: polilíneas, polígonos, líneas dimensionales, texto, etc. No se trabaja con estas entidades a lo largo del presente proyecto, por este motivo no se detallan sus configuraciones.

4.4. Software

4.4.1. Características deseadas del software

Se desea que el software cumpla con las siguientes características:

- Que sea capaz de abrir archivos DXF.
- Que permita identificar las entidades que conforman un tipo especifico de mecanizado.
- Que permita configurar de una manera sencilla e intuitiva los distintos mecanizados a realizar.
- Que posea una variedad de mecanizados a realizar basados en los mas utilizados en la industria.
- Que pueda utilizarse en una amplia variedad de sistemas operativos.
- Que sea fácilmente adaptable a distintas máquinas herramientas.

4.4.2. Librerías preexistentes sobre el tema

Entre las librerias y software preexistente que se encuentran sobre el tema se pueden mencionar:

- YCAD: Libreria para manejo de archivos DXF, desarrollada en JAVA para uso con applets.
- DXF: Software de manejo de archivos DXF, permite crear, abrir y modificar archivos DXF. Desarrollado en JAVA y distribuido bajo licencia GPLv2. Creado en Francia.
- MecaCNC: Software simulador de código G, permite escribir el código y simular su funcionamiento en una fresadora EMCO PC Mill 125. Desarrollado en JAVA de código abierto y distribución libre. Creado en España como tesis de grado de la carrera Ingenieria en Informática.

4.4.3. Lenguajes de programación

Entre las opciones de lenguajes de programación libres que se encuentran en la actualidad, para los fines del presente proyecto se destacan 3: Python, Smalltalk y JAVA. Teniendo en cuenta las librerías y proyectos preexistentes encontrados, se decide investigar a fondo JAVA para tomarlo como lenguaje de programación del proyecto.

4.4.3.1. JAVA:Introducción

JAVA[16]: Es una plataforma para desarrollar aplicaciones. Esta plataforma esta compuesta por al menos 3 componentes:

- El compilador.
- La Máquina Virtual de JAVA (JAVA Virtual Machine - JVM).
- La Interfaz de Programación de Aplicaciones (Application Programming Interface - API)

El primer componente es el encargado de traducir el código que escribimos en bytecodes, lenguaje intermedio entre el código fuente y el lenguaje máquina. La JVM se encarga de ejecutar los bytecodes que generó el compilador. Por último, la API es un conjunto de bibliotecas que nos permiten el desarrollo de las aplicaciones. Estas bibliotecas contienen clases que se utilizan directa o indirectamente en los programas que desarrollemos. Estos componentes vienen incluidos en lo que se conoce como JDK (JAVA Development Kit) o Herramientas de Desarrollo JAVA. El lenguaje JAVA se caracteriza por los siguientes puntos:

- Uso libre: Tanto el compilador como ciertas APIs e IDEs son gratuitas, es decir, no se deben pagar licencias por adquirirlas ni utilizarlas.
- Orientado a Objetos: JAVA se pensó para realizar programación orientada a objetos, cada clase se utiliza como molde para crear objetos.
- Portable: La generación de los bytecodes, independientes de la plataforma, hace que las aplicaciones sean portables, lo que quiere decir que se pueden correr en cualquier máquina que posea la JVM, sin importar el sistema operativo ni la arquitectura.
- Aplicaciones robustas: La exigencia de programar excepciones en ciertos lugares del código hace que las aplicaciones sean robustas. Por robustez se entiende la baja probabilidad que una aplicación tiene de producir fallos o bloqueos.

4.4.4. JAVA: IDE

Integrated Development Enviroment (IDE) o, en castellano, Entorno de Desarrollo Integrado se refiere a las aplicaciones que permiten el desarrollo de nuevas aplicaciones. Teniendo el compilador de JAVA no se necesitaría, en un principio, de un IDE. Se puede escribir el código con cualquier editor de texto. Ese código se guarda como un archivo de extensión .JAVA y, desde la consola se compila. La utilización de un IDE es muy ventajosa ya que los mismos vienen con:

- Un editor de código que nos indica:
  - Con distintos colores si la palabra es reservada o no.
  - Los errores de escritura.
  - Nos despliega todos los métodos que se pueden llamar desde un objeto determinado.
- Un depurador o debugger para seguir el código paso a paso.
- A veces provee un constructor de interfaz gráfica, para aplicaciones que contienen gráficos (como ventanas, botones, etc).

Entre los IDEs mas importantes se encuentran:

- NetBeans
- Eclipse

Ambos son de uso libre.

4.4.5. Comparativa: Eclipse vs NetBeans

Eclipse:

- Más rápido
- Más flexible
- Más plugins
- Mejor soporte para desarrollo Android
- Herramienta Gui SWT. SWT requiere que las bibliotecas nativas se incluyan con el producto final.

Netbeans:

- Más pesado sobre todo teniendo varios proyectos.
- Herramientas Swing estándar.
- GUI más intuitiva.
- SVC más intuitivo (GIT)

- Mejor soporte para PHP
- Permite importar proyectos de Eclipse y otros IDE's

Se decide utilizar Eclipse[17] para abrir la puerta a crear, en un futuro, una versión del software que corra en android, teniendo en cuenta la portabilidad que ello representa.

Capítulo 5

Desarrollo

5.1. Comunicación

Considerando que el software que maneja la fresadora se encuentra actualmente en una PC que corre bajo Windows 95 y ante la imposibilidad de migrar dicho programa a una PC en la que se pudiera correr también el software diseñado en JAVA (el software WINNC que maneja a la fresadora solo corre hasta la versión Millenium (Me) de Windows y las placas necesarias para la comunicación con la fresadora son de arquitectura ISA, actualmente se utiliza arquitectura PCI); se comenzaron a analizar las posibles formas de transmitir los archivos de texto, generados por el software a desarrollar, a la PC de la fresadora en donde se utilizan.
Se revisa la PC para hacer un relevamiento de puertos libres y solo se cuenta con un puerto paralelo. Lamentablemente las PC actuales no cuentan con este puerto por lo que para poder utilizarlo como puerto de comunicación hay que realizar una serie de pasos de adaptación que resultan muy engorrosos.
Debido a que Windows Millenium fue una versión muy inestable de 16 bits, que generalmente funciona con equipos nuevos, y que ante una actualización de software trae aparejado muchos errores, se desestima esta alternativa.
Finalmente se encuentra que la PC posee dos slot PCI libres, por lo que se opta por instalar una placa Ethernet y proceder a la configuración de una red de área local (LAN) entre las dos PC involucradas en el proyecto.

5.1.1. Placa Ethernet instalada

La placa ethernet instalada es una 3com 3c905b-tx-nm, figura 5.1
Características Generales

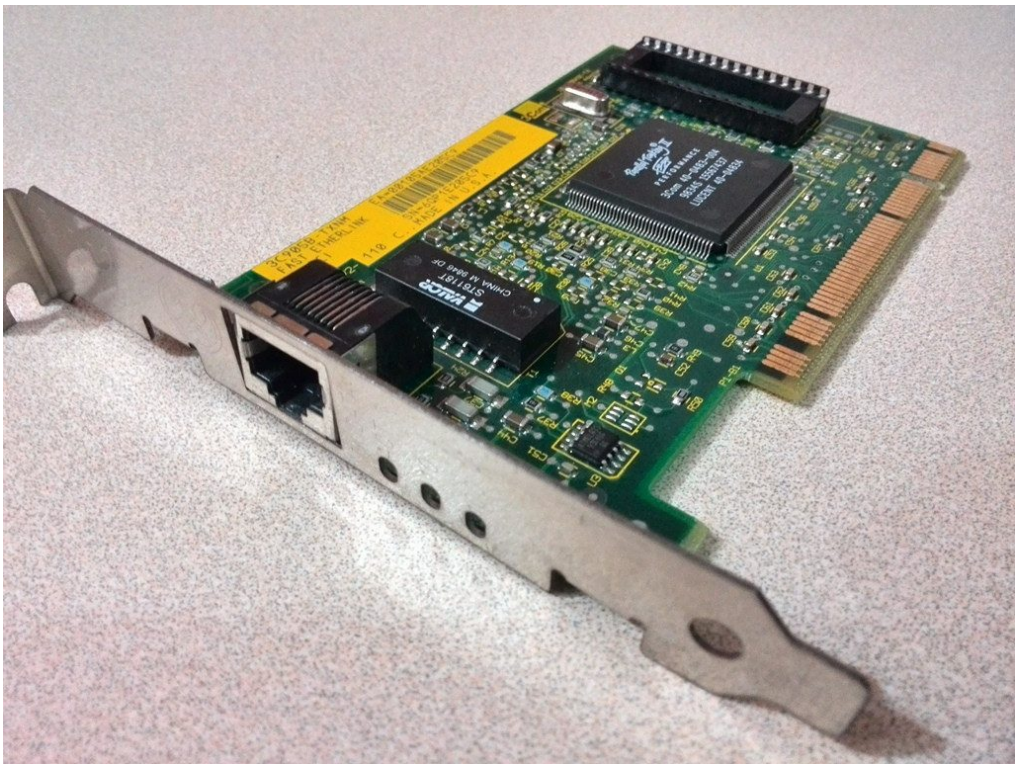


FIGURA 5.1: Placa Ethernet instalada

- velocidad de transferencia de datos: 10/100 Mbit/s;
- Interface: PCI 2.2, 32 bits, 33 MHz;
- Chip: 3Com 3S905B;
- Tamaño de buffer: 4K;

Conexión

- Número de conectores RJ-45: 1;
- Estándares: 802.1p, 802.1Q VLAN, control de flujo 802.3x;

Además

- Soporte para SO: Linux 2.4/2.2, Windows XP/Me/2000/98/NT 4.0, Novell NetWare 5.x;

5.1.2. Detalles de montaje de la red

5.1.2.1. Red entre Windows 95 y Windows 8

Se realiza una investigación sobre el montaje de una red en Windows 95 y finalmente se monta de la siguiente manera:

- Se instala físicamente una placa Ethernet 3com 3c905b-txnm
- Se comprueba que funciona, ya que al iniciar Windows, reconoce el nuevo hardware instalado y procede a pedir los archivos para instalar los drivers.
- Se consiguen los drivers para Windows 95 y con la ayuda de una disquetera USB se graban en disquete, ya que es el único medio posible para ingresar datos a la PC de la fresadora, previo a la instalación de la placa de red.
- Una vez finalizada la instalación de la placa Ethernet se procede a montar y configurar la red de área local (LAN).

Como se quieren conectar solo 2 PC en red, se opta por conectarlas entre sí directamente con un cable Ethernet en vez de utilizar un hub.
No es necesario utilizar un cable cruzado, ya que la placa de la PC donde se realiza el software se encarga de realizar las inversiones de señales.
Se procede a configurar la red. Para esto en la PC con Windows 95 es necesario realizar los siguientes pasos: Una vez instalada la placa de red y el software requerido, se deben instalar y configurar los protocolos de red, necesarios para que los equipos puedan comunicarse entre sí.

- Desde"Mi PC" se accede al "Panel de Control", figura 5.2.

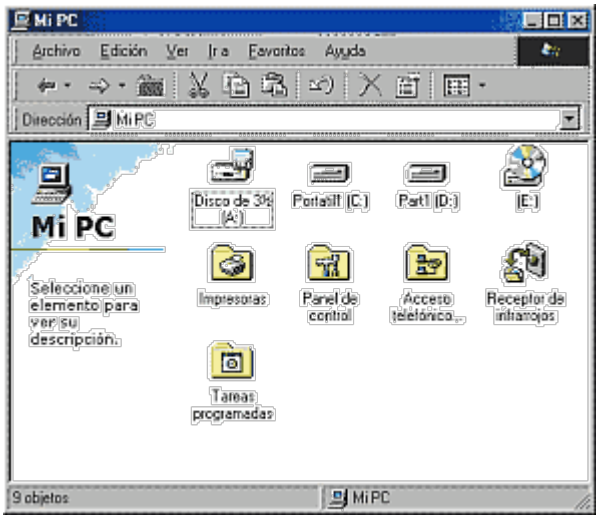


FIGURA 5.2: Windows 95: Mi PC

- Desde el "Panel de Control" se debe acceder a "Red", figura 5.3
- Pulsando en el botón "Agregar", figura 5.4, se deben ir añadiendo los protocolos "TCP/IP" y "NetBEUI"; el servicio "Compartir impresoras y archivos para redes Microsoft", y el "cliente para redes Microsoft". Todos estos protocolos y servicios son necesarios para que las PC se comuniquen entre sí, y con la excepción del TCP/IP, no es necesario configurarlos.

FIGURA 5.3: Windows 95: Panel de Control

- A continuación se debe asignar un nombre al equipo, figura 5.5
- Se configura el protocolo TCP/IP para asignarle una dirección fija, figura 5.6. Al conectar dos equipos entre sí es necesario asegurarse que ambos pertenezcan al mismo dominio.
  Las direcciones IP son cuatro números de 0 a 255 que identifican a los equipos en la red.
  Se debe asignar una dirección IP única a cada equipo. Se Puede comenzar asignando 192.168.0.1 a la primer PC y numerar las demás PC con una IP del tipo 192.168.0.x
  Los pasos son los siguientes:
  - Se accede a las propiedades del protocolo TCP/IP
  - Se asigna la dirección IP 192.168.0.2 y como máscara de subred: 255.255.255.0
  - Se reinicia la PC y desde una ventana de MSDOS se realiza un "Ping", figura 5.7, a la misma PC para asegurarse de que el TCP/IP está bien instalado. También pueden realizarse "pings" a los demás equipos de la red para ver si se recibe respuesta.

En este punto ya se encuentra la red instalada y se puede ver los demás equipos (en este caso uno solo).
Una vez finalizada la configuración en Windows 95, se procede a configurar la PC con Windows 8. En esta PC simplemente se cambia la asignación de IP dinámico a estático y se fija un IP dentro del rango del que se configura en la otra PC como se explica con anterioridad.

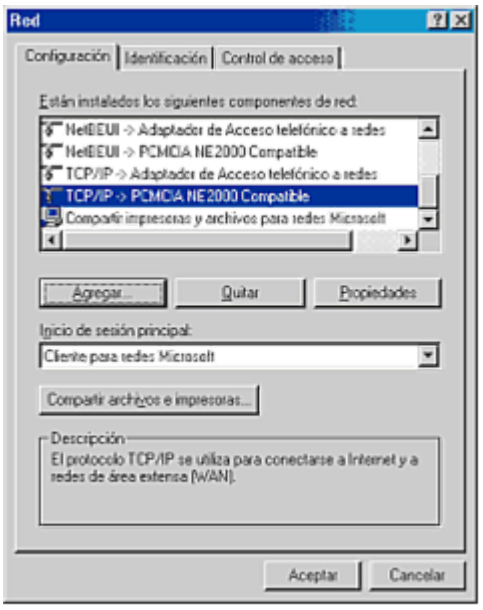


FIGURA 5.4: Windows 95: Red

FIGURA 5.5: Windows 95: Identificación de Equipo

Se revisan las configuraciones de los elementos compartidos y se vuelve a la PC con Windows 95 en donde se comparte la carpeta donde se almacenan los programas que utiliza el controlador Sinumerik 810, con la red.

Otra vez en la PC con Windows 8 se conecta la carpeta compartida como una unidad de red y de esta forma queda establecida la comunicación entre ambas PC.

Se realiza una prueba sencilla copiando archivos de ambas PC a la carpeta compartida y corroborando de esta forma que los archivos pueden abrirse desde ambos sistemas sin inconvenientes.

FIGURA 5.6: Windows 95: Configuración de protocolo TCP/IP

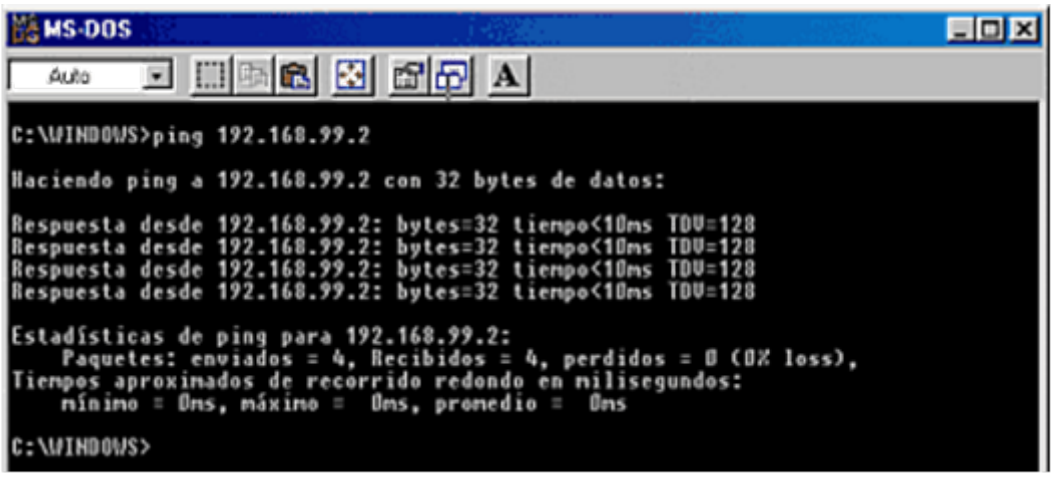


FIGURA 5.7: Windows 95: Ping desde DOS

5.1.2.2. Red entre Windows 95 y Ubuntu 14.04 LTS

Como el presente proyecto pretende desarrollar un software multiplataforma, se considera necesario poder hacer las pruebas del mismo en otros sistemas operativos ademas de Windows. Para eso se decide armar una red de area local entre la PC que controla a la fresadora y una notebook con sistema operativo Linux, especificamente Ubuntu 14.04 LTS.

El primer paso para el montaje de dicha red es configurar el sistema Ubuntu con las mismas caracteristicas antes configuradas en la PC con Windows 8.

Para esto se crea una nueva red cableada en Ubuntu, con los parámetros que se muestran en la figura 5.8. . Una vez configurada la red cableada, se instala SAMBA para poder administrar los recursos

FIGURA 5.8: Parámetros de configuración de Red Cableada en Ubuntu

de red y conectarse de esa forma con una red Windows.

5.2. Proyecto DXF

Teniendo en cuenta lo investigado en el capítulo anterior con respecto a las plataformas de programación utilizables, las librerías y software preexistentes, se elije como plataforma el IDE Eclipse de JAVA, ya que se pretende lograr a futuro una versión del software capaz de correr sobre android. Se utiliza como base el software DXF creado en EPSI París por Stephan Soulard y Edouard Vanhauwaert en el año 2007, distribuido bajo licencia GNU GPL v2. Se utiliza ECLIPSE LUNA v4.4.1 junto con JRE 1.8.0_31 y JDK 1.8.0_31. En la figura 5.9 se observa la pantalla principal del software utilizado como base.

5.2.0.3. Identificación del software DXF

Se realiza una inspección detallada del software DXF a fin de conocerlo en profundidad e identificar los rasgos que son de utilidad para el diseño de DXF2Machine.
Se comienza la inspección haciendo un paneo general de la pantalla principal. Se observa una barra de menues compuesta por dos items: "File"(5.10) y "?". Mas abajo se observa un panel con 4 pestañas entre las que se destacan "Tools"(5.11) y "Statistics"(5.12). Siguiendo para abajo del panel anterior se encuentra la paleta de colores "Color Chooser"(5.13). El último panel ubicado a la izquierda y que se encuentra debajo de la paleta es el Árbol de entidades "Tree View" (5.14). Finalmente en el lado derecho de la pantalla se observa el area de dibujo o "Drawing Area" (5.15).
Como se observa en la figura 5.10, mediante DXF se pueden crear archivos nuevos, abrir y modificar

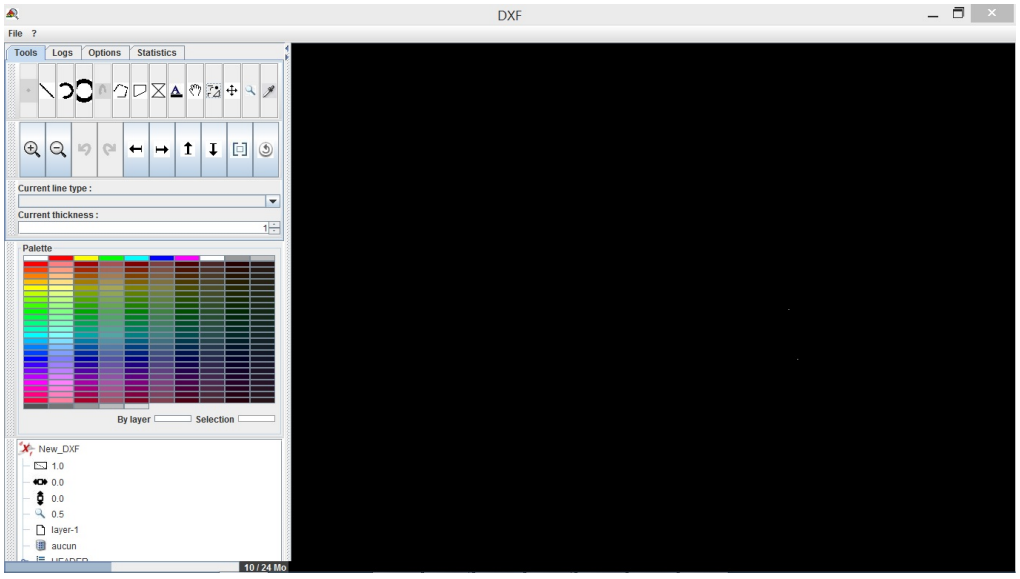


FIGURA 5.9: Pantalla Principal DXF.

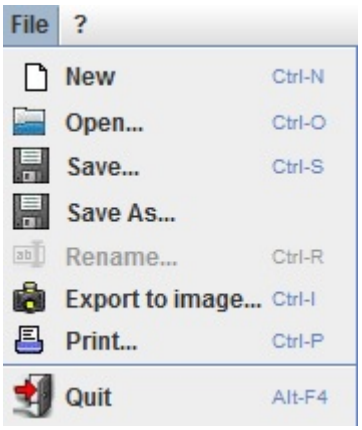


FIGURA 5.10: Menu File

archivos existentes. El software a diseñar pretende manejar archivos .DXF e interactuar con los mismos por lo que en principio se tendría esa parte resuelta mediante la utilización del software base. En la figura 5.11 se distinguen 3 grupos de herramientas:

- Herramientas de diseño: Permiten dibujar entidades. Entre las herramientas disponibles se encuentran: Línea, arco, circunferencia, punto, polilínea, poligono, etc. Lamentablemente su uso es bastante engorroso, no posee forma de acotar y se dificulta enganchar el final de una entidad con el principio de otra. Se opta por prescindir de este grupo de herramientas en el diseño de DXF2Machine. Para las ediciones de archivos .DXF se utiliza Draftsight, un software CAD gratuito diseñado por Dassault Systemes muy completo que cumple con los requerimientos necesarios para el presente proyecto.

- Herramientas útiles: Permiten configurar la visualización del dibujo. Entre las herramientas disponibles se encuentran: Zoom, desplazamientos, Vista centrada, etc. Este grupo se considera

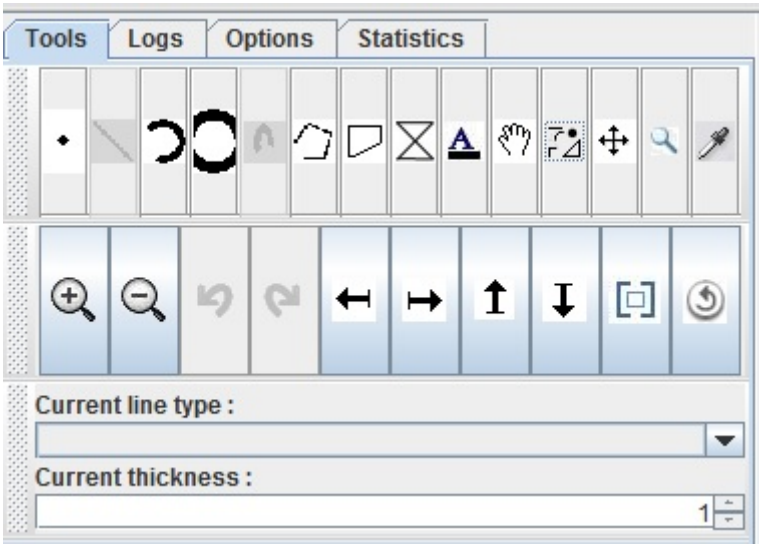


FIGURA 5.11: Panel Herramientas

de utilidad para el proyecto DXF2Machine por lo que se utiliza en el mismo.

- Tipos de línea: Permite elegir el grosor y estilo de línea de la entidad a dibujar. Considerando que no es necesario contar con esta característica para lo que se desea lograr, no se utilizará este grupo de herramientas en DXF2Machine.

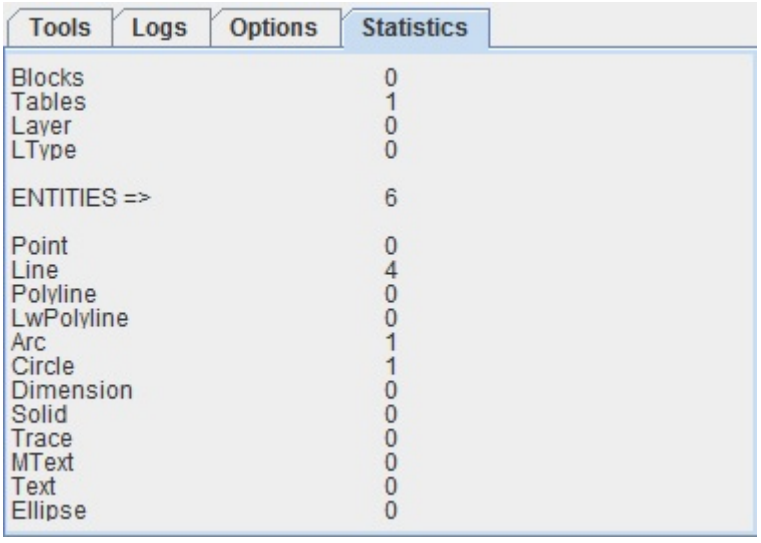


FIGURA 5.12: Panel Estadisticas

La cuarta solapa del panel de herramientas corresponde al cuadro de estadísticas, como se observa en la figura 5.12. Aquí se muestra el recuento de elementos de cada entidad presentes en el dibujo. Si bien esta característica no es de utilidad para el software propuesto, estaría indicando la presencia de una tabla, en donde se guardan todas las entidades presentes en el dibujo. Dato no menor para el diseño de DXF2Machine.

En la figura 5.13 se observa una extensa paleta de colores disponible. Se plantea la idea de utilizar

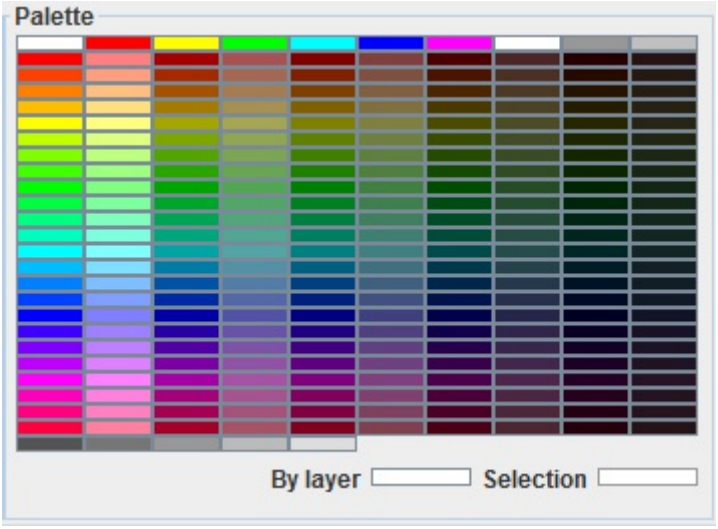


FIGURA 5.13: Paleta de Colores

un código de colores específico para representar los distintos tipos de mecanizados realizables con DXF2Machine, por lo que se mantiene la paleta pero se reduce la cantidad de colores disponibles.

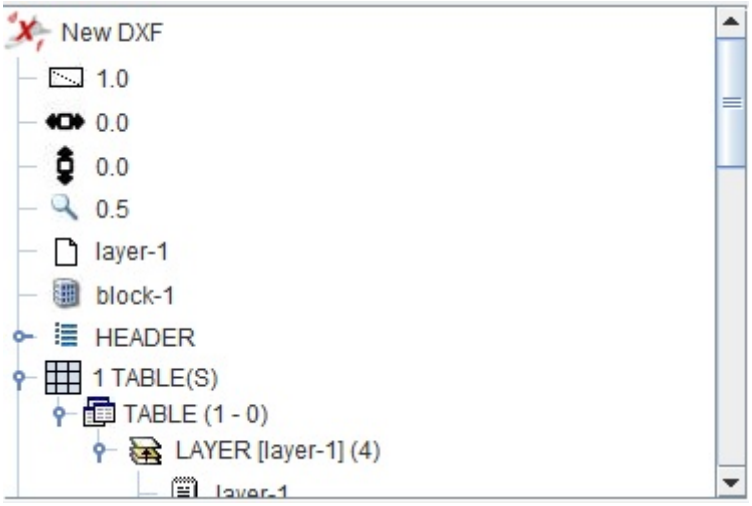


FIGURA 5.14: Árbol de Entidades

El árbol de entidades que se observa en la figura 5.14 confirma la existencia de una tabla en donde se encuentran alojadas todas las entidades y a la cual se accede en el momento de realizar algún cambio sobre las mismas. Desde el árbol se pueden modificar las características de las entidades e incluso eliminarlas y esto se ve reflejado en el panel de dibujo.

El árbol no se incluye en DXF2Machine pero desde aquí se realiza la ingenieria inversa sobre el código para llegar a la tabla y tener acceso a los datos de las entidades necesarios para lograr la traducción del dibujo a código G.

Por último en la figura 5.15 se observa el panel de dibujo, aquí se visualizan los archivos DXF, pueden dibujarse nuevas entidades y modificarse las ya existentes.

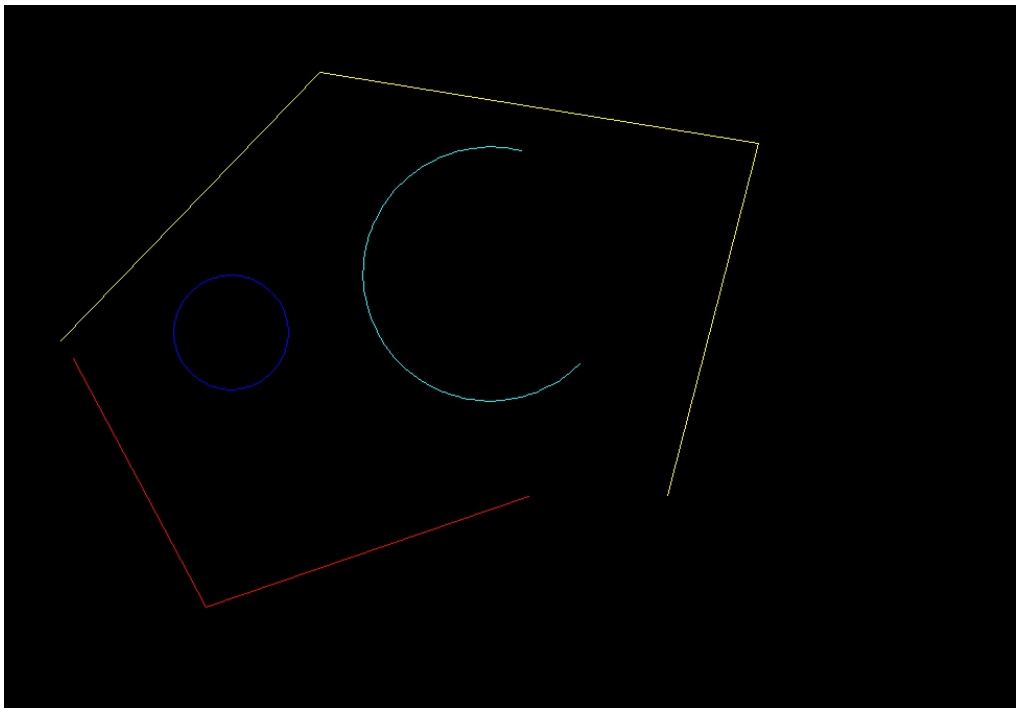


FIGURA 5.15: Panel de Dibujo

Para los fines de DXF2Machine, se modifica el funcionamiento de este panel permitiendo solamente cambiar los colores de las entidades existentes para configurar los mecanizados realizables.

5.3. DXF2Machine: Desarrollo

El desarrollo de DXF2Machine es el resultado de los siguientes pasos:

5.3.1. Obtención de datos de las entidades dibujadas

Para poder generar el código G se necesita conocer las coordenadas de las entidades seleccionadas y sus características, ya que existen codificaciones particulares para distintas geometrías. Por ejemplo se utiliza G01 para indicar que se genera un movimiento lineal y G02/G03 para indicar un movimiento circular.
Del analisis del proyecto DXF surge que los datos de las entidades se almacenan en una tabla que se actualiza al modificar cualquier valor de una entidad. Por lo que se decide tomar las entidades de dicha tabla y generar una nueva con los datos necesarios para traducir las entidades a código máquina.

5.3.2. Definición de entidades permitidas en la primer versión del software

Los archivos DXF pueden formarse por una gran variedad de entidades de mayor o menor complejidad. Para esta primer versión de DXF2Machine se trabaja con líneas, arcos y circunferencias. Estas entidades permiten crear un sin fin de geometrías para generar piezas básicas que se utilizan en la industria. Sabiendo que se utilizarán estos 3 tipos de entidades, se analizan los datos guardados en cada una de ellas para saber si son suficientes o se necesitan datos extras para la traducción. Viendo que en los 3 casos puede prescindirse de ciertos datos almacenados y es necesario calcular otros que no están contemplados, se decide crear la clase "Datos", que a su vez tiene 3 subclases como se muestra en la figura 5.16 y cada una de ellas se encarga de manejar los datos necesarios para lograr luego la traducción.

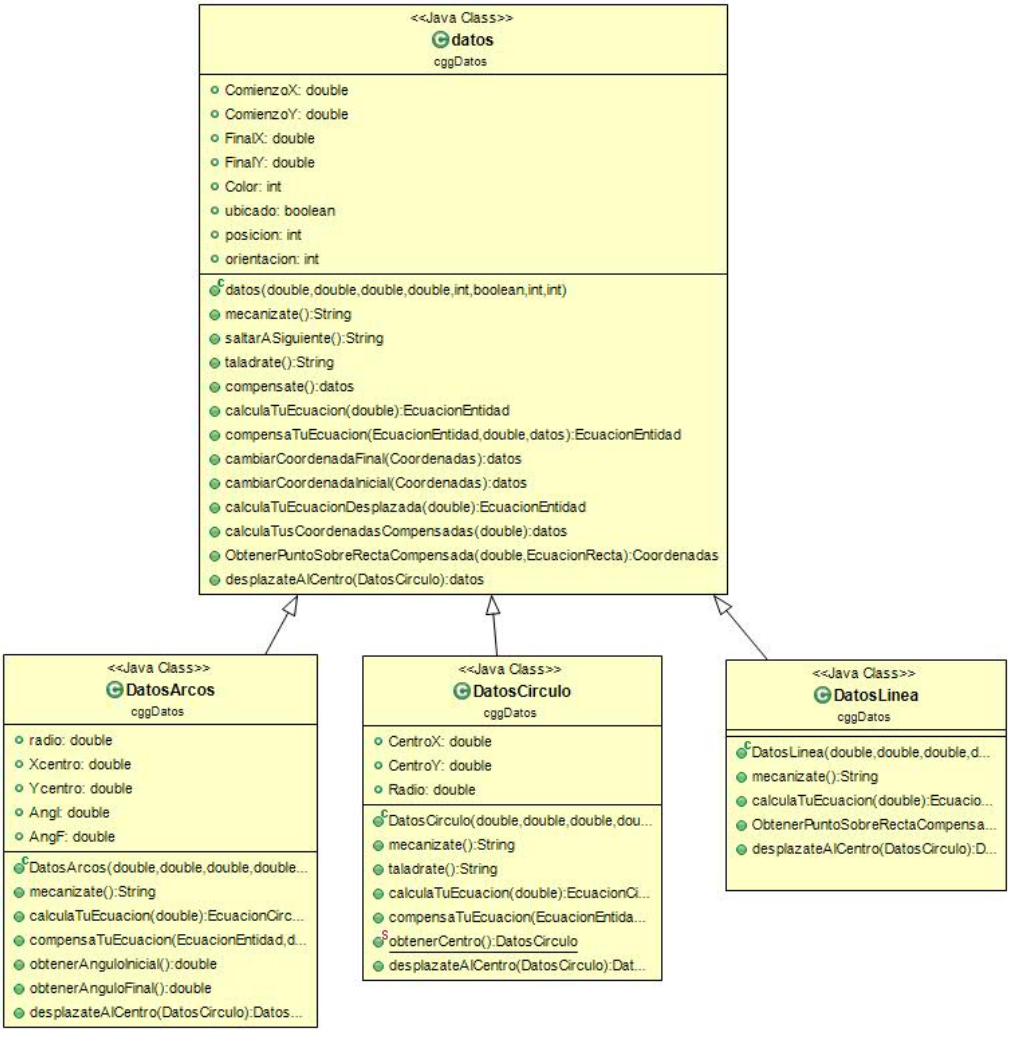


FIGURA 5.16: Estructura de la clase "Datos"

5.3.3. Definición de los mecanizados realizables

El mecanizado de una pieza consiste de varias operaciones que, como se dijo en el capitulo 4, pueden dividirse entre operaciones de preparación y terminación de una pieza.
Para poder llevar a cabo las distintas operaciones, los software CAM definen una serie de "rasgos" mecanizables que se diferencian entre si por las características de sus trayectorias o las herramientas a utilizar. Para DXF2Machine se decide utilizar como rasgos mecanizables: planeado, contorneado, grabado y taladrado.
Planeado se refiere a la preparación superficial del material para lograr su planitud con respecto a una superficie que ya cuente con esta condición, en este caso la mesa o la morsa de la máquina a utilizar.
Contorneado se refiere al perimetro exterior de la pieza dibujada. Se recorre dicho perimetro con una herramienta que transforma en viruta el excedente de material entre la pieza terminada y el "tocho" o material de partida. Para esto es necesario realizar compensaciones de radio de herramienta ya que en un CNC la trayectoria definida es la que sigue el centro de la herramienta. Si no se realizara compensación se tendría un error en la geometría igual al diametro de la herramienta en la medida final tanto en largo como en ancho.
Grabado se refiere a tallar una trayectoria determinada en la superficie de un material. La trayectoria puede tener discontinuidades y líneas entrecruzadas.
Taladrado se refiere al proceso de agujereado de un material.

5.3.4. Definición del método de selección de los mecanizados a realizar

Una vez definidos los rasgos mecanizables, es necesario indicar que entidades corresponden a que tipo de mecanizado. Para ello se decide usar un código de colores, aprovechando la característica de capas de los software CAD que definen distintas partes de un mismo plano, utilizando distintas capas con distintos colores.
Se propone utilizar el siquiente código de colores:

- Rojo: para definir el centro de referencia de la pieza a mecanizar.
- Azul: para definir el contorno de la pieza a mecanizar.
- Celeste: para definir las entidades que componen el grabado.
- Verde: para definir los agujeros que se realizan mediante taladrado.

Como cada entidad originalmente cuenta con el dato de su color, se utiliza este dato para separar los elementos en distintas tablas según el tipo de mecanizado que componen.
De esta forma se crea el paquete "cggTablas" que agrupa las clases de tablas que manejará DXF2Machine.

5.4. Diseño de la interfaz para DXF2Machine

Se decide utilizar como base la interfaz del proyecto DXF agregando y/o quitando herramientas para definir el alcance del proyecto.
Se actualiza el menú "file" y "?" con los cambios necesarios para adecuarlos al nuevo proyecto:
Se realizan modificaciones en el "acerca de..." y se elimina la opción de "crear nuevo" archivo.
Se reemplazan las pestañas originales que se muestran en la figura 5.11 (Tools, Logs, Options, Statistics) por las pestañas Configuraciones, Herramientas y GCode.

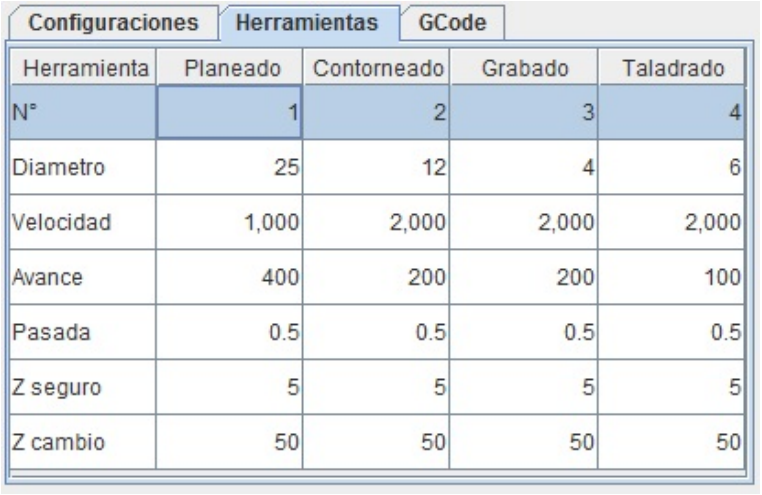
5.4.1. Configuraciones

Se toma la barra de visualización del proyecto DXF y se añade una paleta de colores modificada que permite seleccionar un color y "pintar" las entidades mediante un click. Esta pestaña puede observarse en la figura 5.17.

FIGURA 5.17: Pestaña Configuraciones

5.4.2. Herramientas

Aquí se cargan los datos de las herramientas que se utilizan en el mecanizado de cada rasgo seleccionado en la pestaña anterior.
Todos los datos configurables se muestran en la figura 5.18.
Las características de la herramienta para cada operación son esenciales para la traducción ya que sin estos datos no puede llevarse a cabo el mecanizado.



Herramienta	Planeado	Contorneado	Grabado	Taladrado
N°	1	2	3	4
Diametro	25	12	4	6
Velocidad	1,000	2,000	2,000	2,000
Avance	400	200	200	100
Pasada	0.5	0.5	0.5	0.5
Z seguro	5	5	5	5
Z cambio	50	50	50	50

FIGURA 5.18: Pestaña Herramientas

5.4.3. GCode

Esta pestaña, como se muestra en la figura 5.19, permite seleccionar la máquina para la cual se lleva a cabo la traducción, y también los rasgos que se desean mecanizar.
Por último el botón "Generar Código" da la orden de realizar la traducción y grabar el resultado en el o los archivos según corresponda.

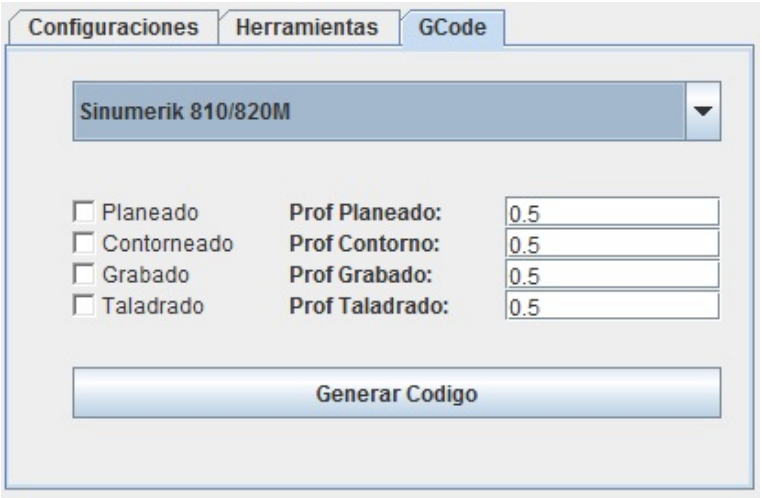


FIGURA 5.19: Pestaña GCode

5.4.4. Consola

Además de las pestañas antes mencionadas para configurar la traducción, se agrega una "consola", la cual se muestra en la figura 5.20, en donde se puede ver el código generado por la traducción. En dicha consola se especifica el código principal y cada uno de los códigos generados para los dis-

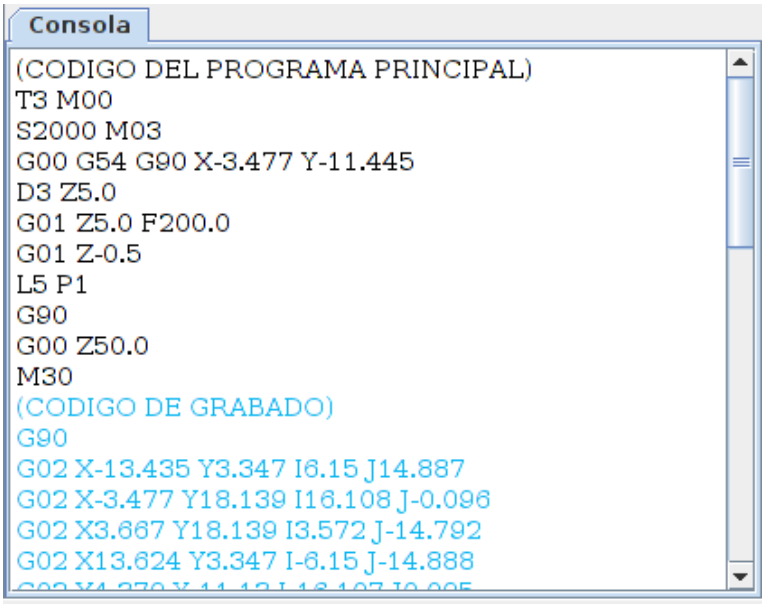


FIGURA 5.20: Consola

tintos rasgos mecanizables seleccionados.
Para que la identificación del código correspondiente a cada rasgo sea mas visual se decide mostrar el texto de cada rasgo con su correspondiente color de selección.

5.4.5. Area de Dibujo

Se mantiene del proyecto DXF el área de dibujo (figura5.21) en donde se visualizan los archivos DXF y se puede interactuar con los mismos mediante el cursor para colorear los distintos rasgos.

5.5. Algoritmos que componen DXF2Machine

Teniendo armada la interfaz gráfica, es momento de desarrollar los algoritmos mediante los cuales, al presionar el boton “Generar Código”, se genera la traducción y se graba en uno o mas archivos compatibles con la máquina en cuestión.

5.5.1. Algoritmo General de Traducción

En la figura 5.22 se muestra el proceso general de traducción.
El código del proyecto DXF2Machine y las modificaciones realizadas al proyecto DXF para su integración, pueden observarse en anexo. A continuación se describen cada uno de los pasos necesarios para lograr la traducción. Se pone especial énfasis en los mas importantes:

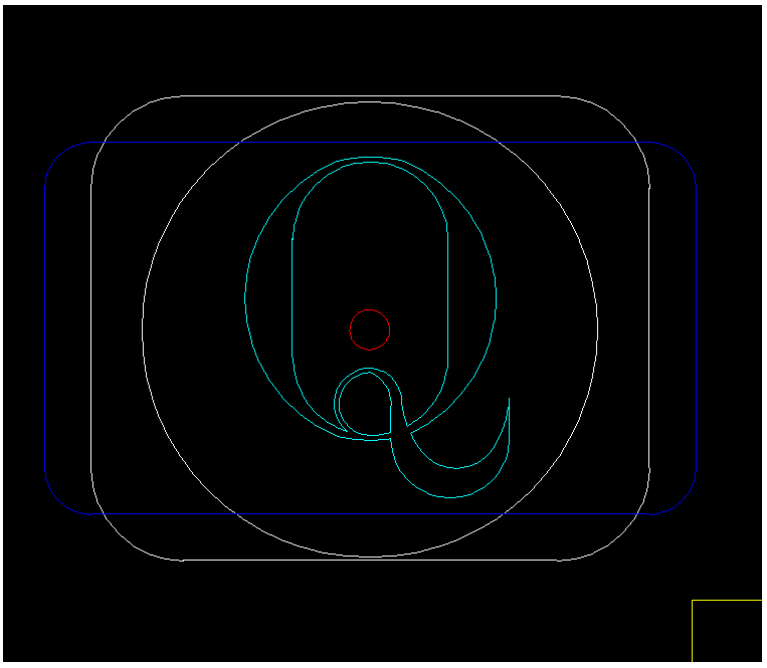


FIGURA 5.21: Area de Dibujo heredada del proyecto DXF

- Obtener tabla entidades: Se recorre la tabla original de entidades que genera el proyecto DXF y se toman los datos de las entidades permitidas para generar una tabla de datos.
- Clasificar entidades por tipo de mecanizado: La tabla obtenida en el paso anterior posee los datos de todas las entidades sin estar agrupadas por rasgo mecanizable. Se recorre esta tabla y se generan tablas independientes para cada rasgo clasificando los datos.
- Preparar PostProcesador: Este algoritmo carga un archivo .properties con toda la información de configuración de la máquina especificada. Esto es fundamental ya que cada máquina posee una forma particular de nombrar el archivo, encabezar el código e incluso de nombrar y configurar las instrucciones necesarias para llevar a cabo el mecanizado.
- Encabezar programa principal: Cabe aclarar que la mayoría de las maquinas CNC permiten generar todo el código en un solo archivo o tener un programa principal y varios subprogramas. Los subprogramas pueden estar incluidos en el mismo archivo (subrutinas) o en archivos separados.

  DXF2Machine permite, tanto generar un único archivo conteniendo la totalidad del código generado, como también generar un archivo correspondiente al programa principal y archivos separados para los subprogramas.

  En el caso de que se utilice un solo archivo, se considera que ese es el programa principal. Por lo tanto en ambos casos es necesario encabezar el archivo.

  En el encabezado se incluyen funciones preparatorias y de seguridad (cancelación de ciclos fijos, configuración de movimientos en coordenadas absolutas, selección de las coordenadas de referencia, etc).

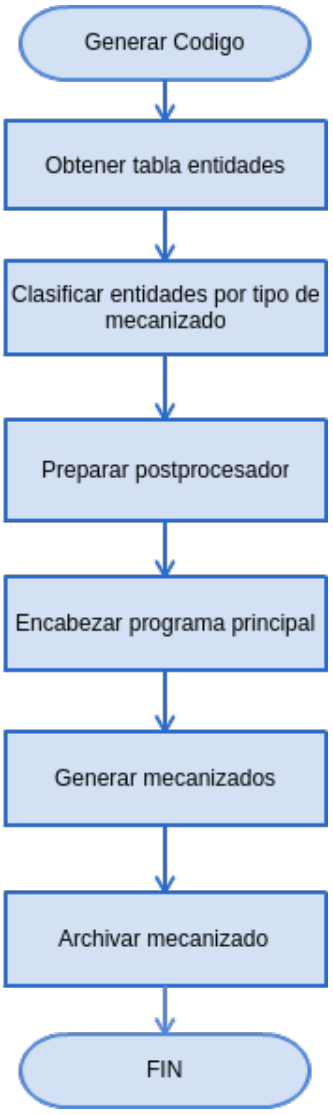


FIGURA 5.22: Diagrama de Flujo de la Generacion de Codigo

- Generar Mecanizados: Una vez cumplidos los pasos anteriores se realiza el chequeo de los mecanizados, que se deben realizar en un orden predeterminado. Este orden no es casual, se debe al orden lógico que se sigue en la industria para el proceso de mecanizado. El algoritmo de generación de mecanizados se muestra en la figura 5.23.
- Archivar Mecanizado: Una vez generadas las traducciones de todos los rasgos seleccionados, se generan los archivos de salida.

5.5.2. Generar Planeado

El algoritmo encargado de generar el planeado (figura 5.24), no solo debe realizar una traducción, sino que también debe encargarse de generar la tabla con la trayectoria a seguir para realizar el mecanizado. Los pasos para generar el planeado son:

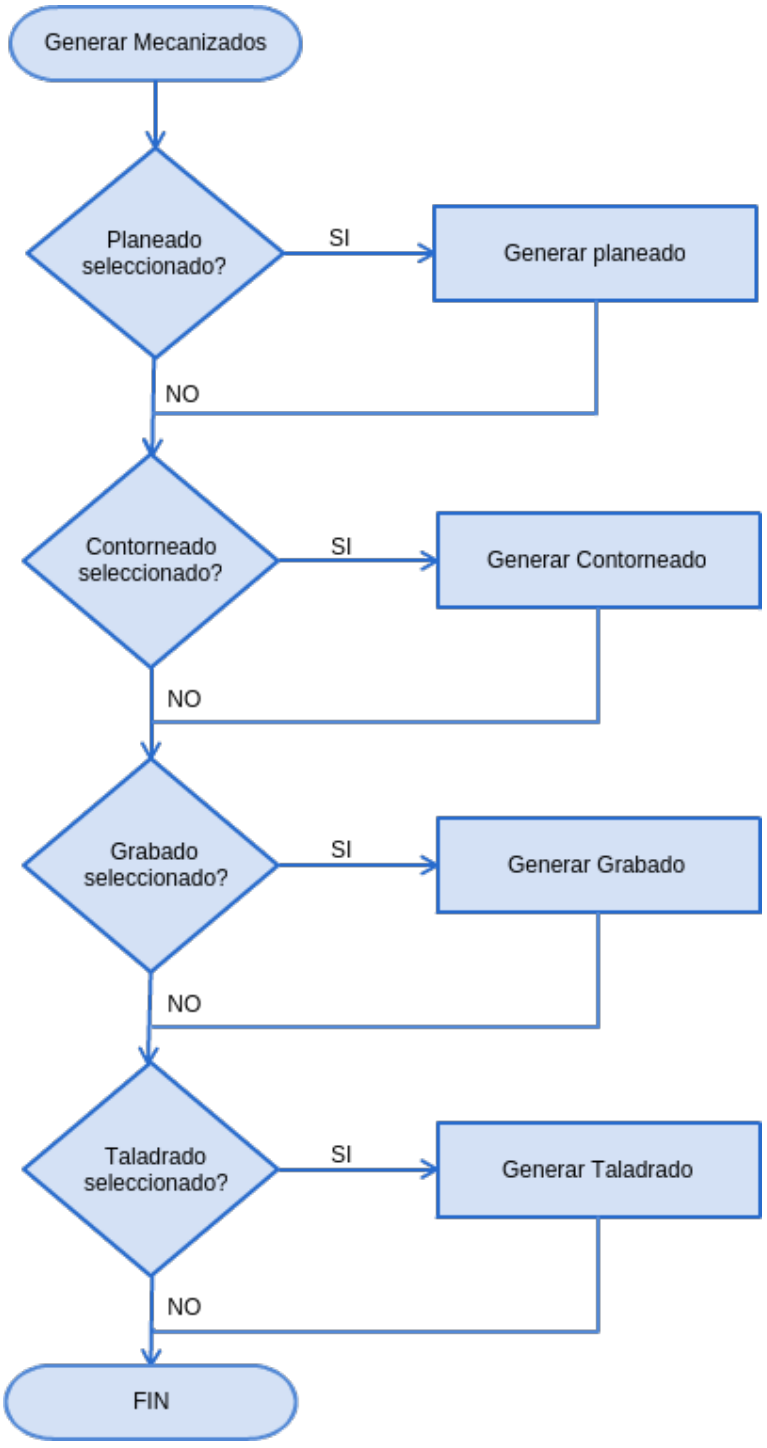


FIGURA 5.23: Algoritmo para generación de mecanizados

- Inicializar mecanizado: Se configuran los parámetros de herramienta tales como número de la misma, velocidad, avance, compensaciones y sistema de referencia.
- Generar Tabla Planeado: La trayectoria se calcula a partir de la tabla de entidades correspondiente al contorneado. Se busca el mínimo valor de X e Y, luego el máximo, y con estos datos

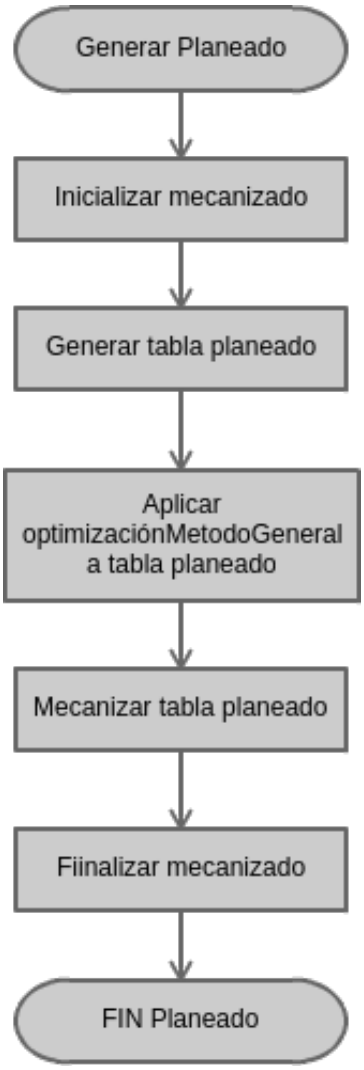


FIGURA 5.24: Algoritmo para generación del planeado

mas el diámetro de la herramienta a utilizar se genera una trayectoria capaz de barrer toda la superficie entre dichos puntos.

- Aplicar optimización a tabla planeado: La optimización se encarga de revisar la tabla de entidades y, de ser necesario, aplicar un criterio sobre la misma para que el mecanizado que se genere cumpla con determinados requisitos tales como: que se realice en el menor tiempo posible, buscar las entidades mas cercanas para reducir el tiempo de desplazamiento entre una y otra, etc. En el planeado, la tabla ya se genera de forma optima por lo que la optimización devuelve la misma tabla. En la figura 5.25 se muestra el diagrama de flujo del algoritmo responsable de la optimización.

- Mecanizar tabla planeado: Este algoritmo realiza la traducción propiamente dicha. Toma las entidades de la tabla de a una en orden, analiza de que entidad se trata, toma los valores necesarios y genera la traducción de esa entidad en particular. Luego toma la siguiente y realiza el mismo proceso. Asi sucesivamente hasta llegar al final de la tabla.

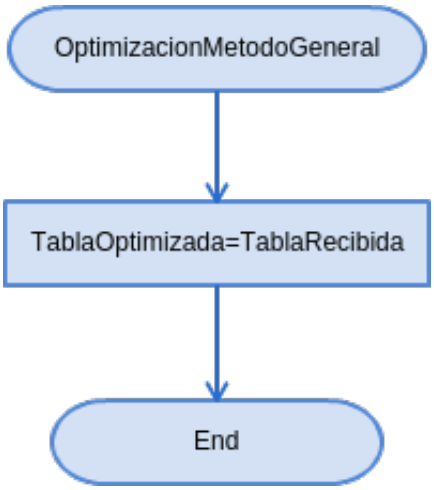


FIGURA 5.25: Algoritmo para optimización segun método general

- Finalizar mecanizado: Una vez realizada la traducción, se configuran los parámetros necesarios para indicar que se concluyó esa operacion. Para esto en el caso particular de las maquinas CNC, se mueven los ejes a una posición de seguridad.

5.5.3. Generar Contorneado

El contorneado (figura5.26) es el algoritmo que mayor dificultad representa. Para poder llevarse a cabo, se deben realizar una serie de chequeos y transformaciones a la tabla de entidades y estos pasos son de gran complejidad.

- Inicializar mecanizado: idem al explicado para el planeado.
- Aplicar OptimizacionMetodo2 a tabla Contorneado: Esta optimización se encarga de ordenar las entidades de la tabla, chequear que se trate de un contorno cerrado, orientar los elementos y compensar la trayectoria teniendo en cuenta el diametro de la herramienta utilizada en este mecanizado. En la figura 5.27 se muestra el diagrama de flujo del algoritmo responsable de la optimización.
  Es importante destacar que si no se tratase de un contorno cerrado o no se pudiera lograr una trayectoria compensada con la herramienta seleccionada, el mecanizado NO se genera.
- Mecanizar tabla contorneado: Idem al explicado para el planeado.
- Finalizar mecanizado: Idem al explicado para el planeado.

5.5.4. Generar Grabado

El grabado (figura 5.28)permite discontinuidad entre las distintas entidades que lo componen. Esta particularidad hace necesario que se realice un chequeo entre la entidad actual y la siguiente para

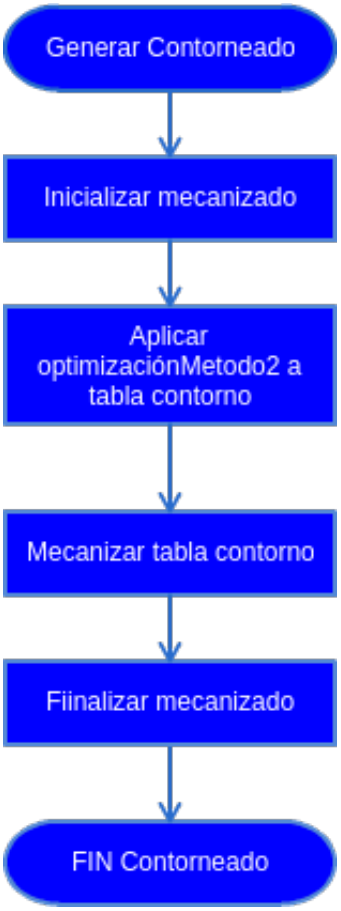


FIGURA 5.26: Algoritmo para generación del contorneado

saber si se debe realizar un salto entre ambas entidades o si en cambio se presenta una continuidad entre ellas.

- Inicializar mecanizado: idem al explicado para el planeado.
- Aplicar OptimizacionMetodo1 a tabla Grabado: Esta optimización se encarga de ordenar las entidades de la tabla buscando todas las que se encuentran conectadas y minimizando la distancia entre ellas al momento de producirse una discontinuidad en la trayectoria. En la figura 5.29 se muestra el diagrama de flujo correspondiente a la optimización.
- Mecanizar tabla grabado: Idem al explicado para el planeado.
- Finalizar mecanizado: Idem al explicado para el planeado.

5.5.5. Generar Taladrado

El taladrado (figura 5.30) es un caso particular en el mecanizado, ya que la mayoría de las máquinas vienen preparadas con lo que se conoce como “ciclos fijos” entre los que se encuentran los ciclos para taladrado y taladrado profundo.

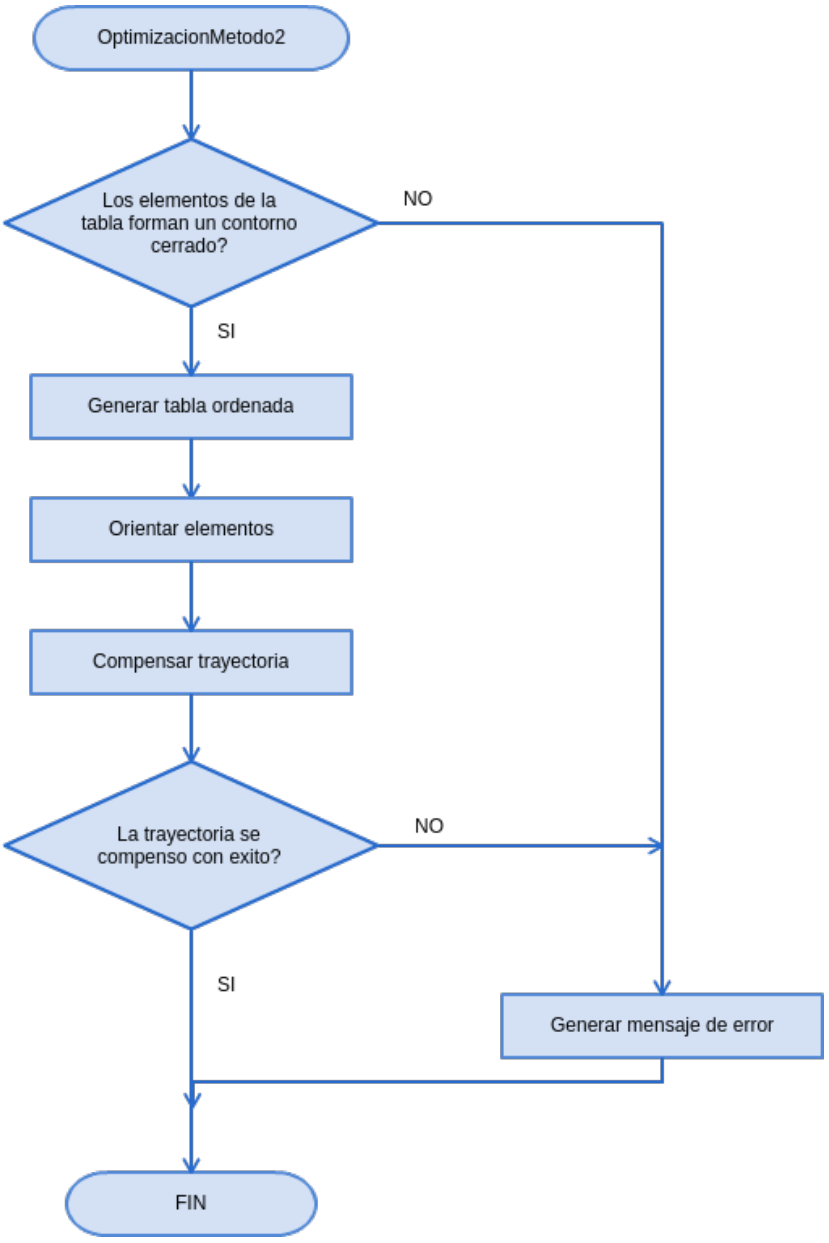


FIGURA 5.27: Algoritmo para optimización según método 2

- Inicializar mecanizado: Se configuran los parámetros para la inicialización del ciclo fijo correspondiente. Si el agujero a realizar es profundo, generalmente se realizan descargas de viruta cada una cierta cantidad de milimetros. Si por el contrario el agujero no es profundo o el material lo permite, el agujero se realiza de una sola vez.
- Aplicar Optimización a tabla Taladrado: Se utiliza la misma optimización que en el planeado.
- Mecanizar tabla taladrado: Se listan los centros de todos los agujeros a realizar.
- Finalizar mecanizado: Se cancela el ciclo fijo utilizado.

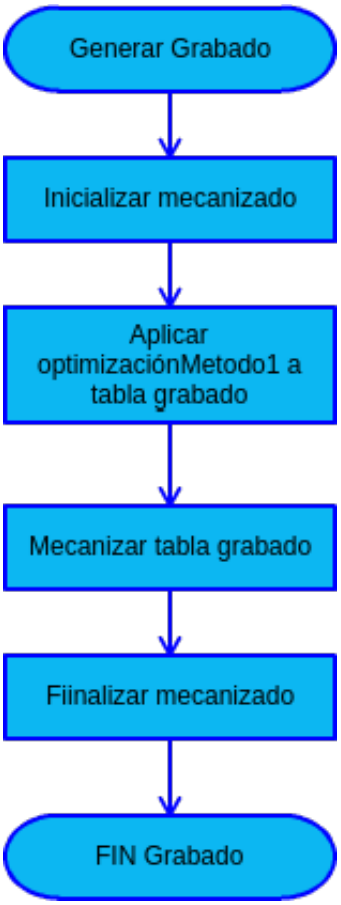


FIGURA 5.28: Algoritmo para generación del grabado

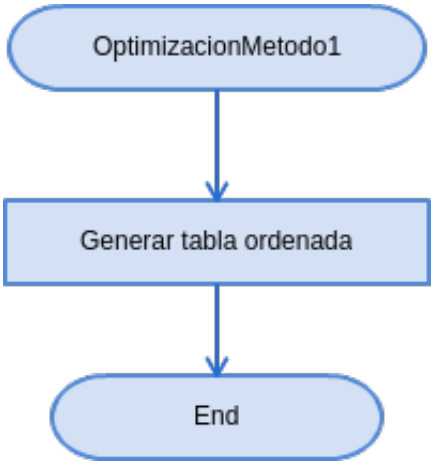


FIGURA 5.29: Algoritmo para optimización según método 1

5.5.6. Configuración de las máquinas compatibles con la primer versión

En anexo se encuentran los archivos de configuracion de las dos máquinas compatibles con la primer versión de DXF2Machine.

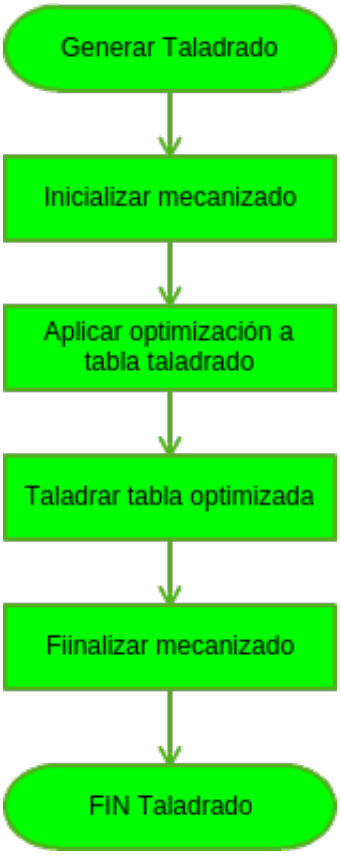


FIGURA 5.30: Algoritmo para generación del planeado

5.6. Generación de archivos con el código resultante

Para la primer versión de DXF2Machine se decide no permitir elegir el nombre de los archivos. El programa principal se guarda con la estructura de nombre particular de cada máquina mas un número y los subprogramas se guardan de la misma manera. Se realiza un chequeo para comprobar si el archivo no existe, en el caso de que el archivo a crear ya exista, se incrementa el número propuesto como nombre del programa y se vuelve a chequear. Este procedimiento se repite hasta encontrar un nombre que no pertenezca a ningun archivo preexistente.

Se permite elegir la ubicación en donde se guardaran los archivos, en Windows pueden guardarse en cualquier carpeta, unidad extraible o sitio de red. Los sistemas Unix, en cambio, solo permiten seleccionar carpetas del sistema. Por lo tanto es posible guardar en cualquier carpeta o unidades extraibles siempre y cuando esten montadas en alguna carpeta del sistema.

Al generar con éxito la traducción, en la consola se puede observar el código de cada archivo generado. Junto con el código del programa principal se observa la información de ubicación y nombre del archivo en el cual se grabo.

Capítulo 6

Pruebas

6.1. Comunicación

Se relizan varias pruebas para comprobar la comunicación, tanto en la Red montada con Windows como con la red montada con Ubuntu.

6.1.1. Pruebas en red Windows

La primer prueba y más básica consta en realizar desde la consola de ambas PC un PING hacia la otra PC para asegurarse de tener respuesta. La prueba es satisfactoria. Luego se comparten carpetas entre ambas maquinas y se intenta guardar archivos de ambas PC y editarlos tambien desde ambas PC. La prueba una vez más es satisfactoria.

6.1.2. Pruebas en red Ubuntu

Se conecta la red y se realizan una serie de pruebas.
La primer prueba y más básica, idem a la realizada en Windows, es realizar un PING entre ambos sistemas. Se obtiene resultado satisfactorio, comprobando de esta manera que la conexión se ha establecido con exito.
Se monta la carpeta en donde se quieren grabar los archivos producto de la traducción en la PC con Ubuntu. Se debe tener en cuenta que como el software WinNC levanta los archivos de una carpeta especifica y no permite seleccionar otra, se decide compartir esta carpeta en la red para grabar los archivos generados directamente en esa ubicación.
Se realizan copias de archivos desde ambos sistemas y se observa que los mismos pueden ser accedidos por ambas PC.

Habiendo realizado las pruebas anteriores con exito se procede a enviar un archivo de código directamente desde DXF2Machine a la carpeta especifica de EMCO. Aqui surge un problema: Los sistemas Unix no permiten acceder a ubicaciones fuera del sistema de archivos propio desde JFileChooser u otro selector de archivos de JAVA. Se plantea la posibilidad de modificar el codigo de DXF2Machine para poder seleccionar otra ubicacion pero luego de investigar se desiste de esta idea por parecer infructuosa.
Se intenta mediante el uso de SAMBA montar la carpeta remota en una carpeta propia de Ubuntu, pero lamentablemente Windows 95 limita esta opción y no es posible lograrlo.
Finalmente desde Windows 95 se conecta como unidad de red una carpeta compartida en Ubuntu logrando asi establecer la comunicación. Para terminar la configuración se procede a cambiar la ruta de la carpeta donde EMCO busca los archivos de código, este último paso puede observarse en la figura 6.1. Se realiza una prueba con esta nueva configuración, para ello se graba un archivo gene-

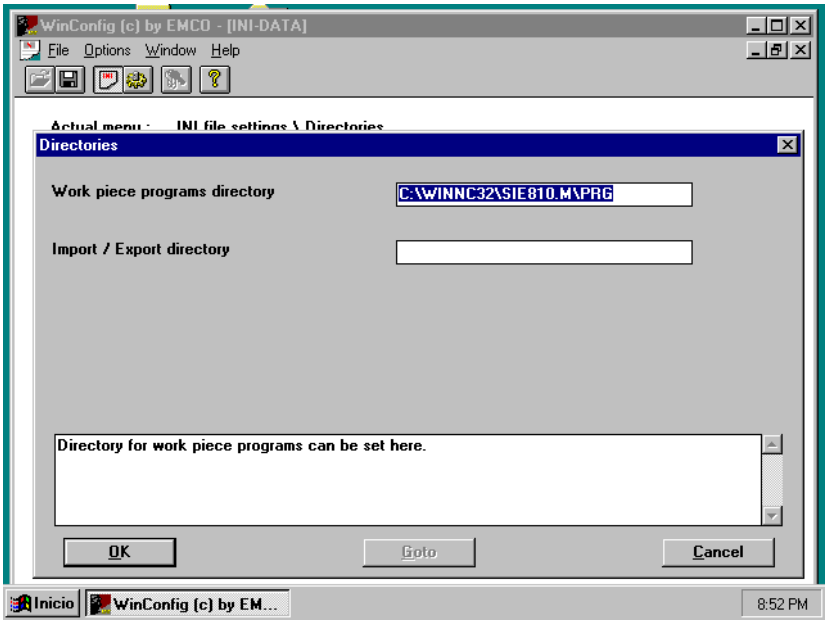


FIGURA 6.1: Programa de configuración de WinNC

rado desde DXF2Machine en la carpeta compartida en el sistema de archivos de Ubuntu. Se abre el software WinNC y se intenta acceder al archivo generado. La prueba es satisfactoria.
En la figura 6.2 puede verse la unidad de red y los equipos conectados desde Windows.

6.2. Software

6.2.1. Pruebas con controlador Sinumerik 810 en fresadora EMCO PC Mill 55

Se realizan en una primer etapa pruebas de simulación del mecanizado luego, habiendo chequeado el correcto funcionamiento del código generado, se realizan pruebas de mecanizado en aluminio y

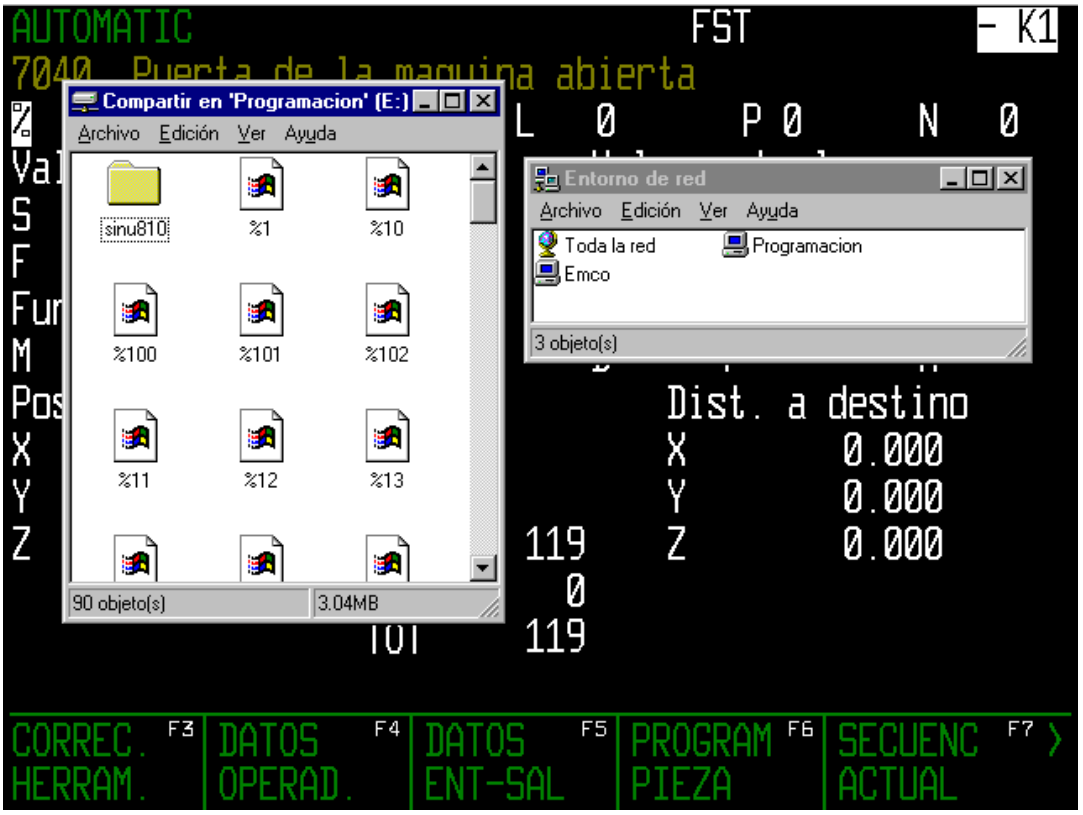


FIGURA 6.2: Visualización de equipos que componen la red desde Windows 95

acrílico. Se realizan distintos archivos en DraftSight y en SolidWorks que si bien es un software CAD 3D en el que se dibujan solidos de forma paramétrica, permite obtener los planos de los sólidos generados y convertirlos a formato DXF. Se comprueba que en ambos casos los archivos pueden leerse con DXF2Machine y puede generarse el mecanizado, siempre y cuando se cuente con las entidades permitidas y los elementos básicos para la configuración de centrado. Para las pruebas tanto de simulación como de mecanizado se utilizan principalmente dos archivos que pueden observarse en las figuras 6.3 y 6.4

6.2.1.1. Pruebas de simulación

Para realizar las pruebas de simulación, se generan los mecanizados deseados con ambos archivos de prueba. Se envian los datos generados a la fresadora EMCO PC MILL 55, obteniendo los siguientes resultados:

- 1er prueba: La simulación se realiza con exito, pero se deben ajustar los parámetros de movimiento de ejes entre mecanizados ya que se nota que la herramienta no sube entre el final de un mecanizado y el comienzo del siguiente, mecanizando trayectorias no deseadas.
- 2da prueba: Habiendo corregido los errores obtenidos en la primer prueba se realiza una nueva simulación. La prueba es exitosa.

FIGURA 6.3: Archivo DXF de prueba 1

En las figuras 6.5 y 6.6 se muestran las simulaciones obtenidas para los archivos de prueba 1 y 2.

6.2.1.2. Pruebas de mecanizado

Habiendo realizado las simulaciones con exito se procede a realizar pruebas en materiales.
La primer prueba se realiza con la configuración de herramientas que se muestra en la figura 6.7 Se utiliza un bloque de aluminio y se generan los 4 tipos de mecanizados posibles utilizando el archivo de prueba 1.
Se toma como decalaje de origen el centro del material y se configuran las alturas de las herramientas en la tabla correspondiente del software WinNC. En la figura 6.8 se muestra la pantalla de configuración de una herramienta.
Se visualiza el programa deseado como puede verse en la figura 6.9. Una vez chequeado que se trata del programa deseado se procede al mecanizado.
Con todo configurado, y el material sujeto en la morsa de la fresadora, se procede a comenzar la prueba. Durante la misma se puede observar en pantalla la línea actual de código a realizar y otros parámetros útiles como se muestra en la figura 6.10
En las figuras 6.11 y 6.12 se puede ver la prueba en proceso y la pieza terminada respectivamente.
La prueba fue satisfactoria, no obstante surgieron algunas puntos a mejorar:

- Se corrige el metodo de contorneado cambiando su método de entrada para evitar que la fresa entre de lleno sobre el material.

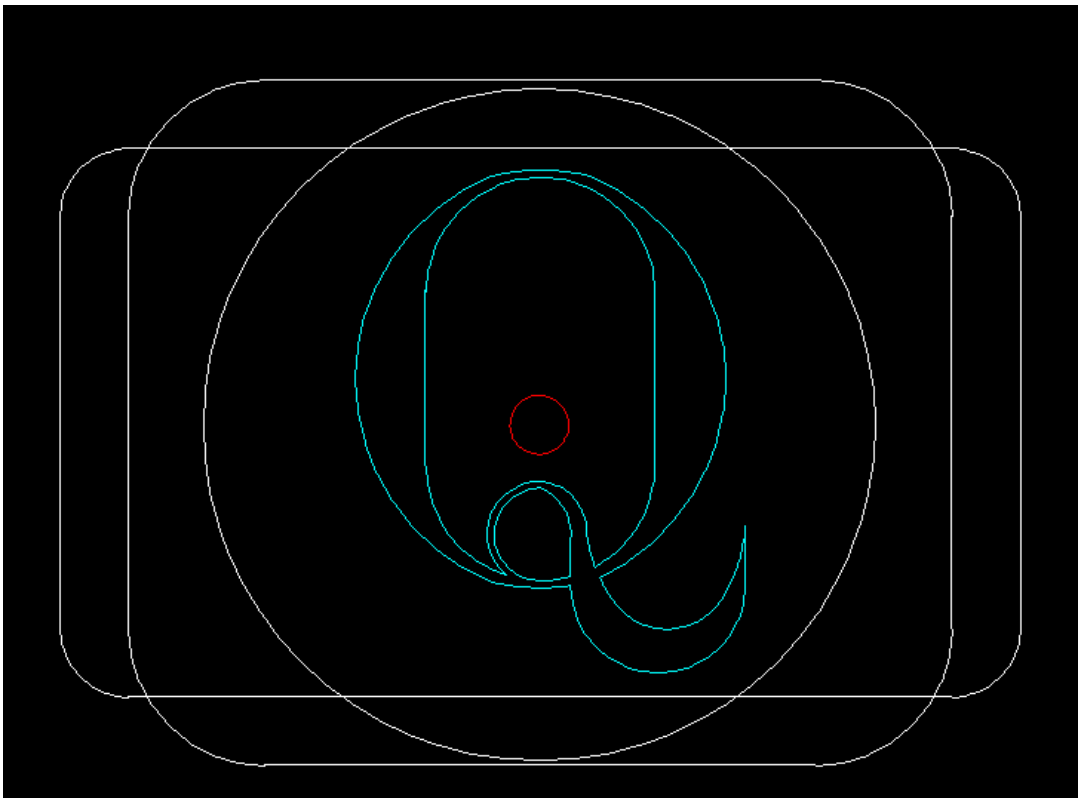


FIGURA 6.4: Archivo DXF de prueba 2

- Se corrige la referencia que se toma en Z, como plano de seguridad en los saltos por discontinuidad de trayectoria, ya que la herramienta subia demasiado y se perdia mucho tiempo en el mecanizado.

Se repite la prueba luego de los cambios. Como resultado se observa un error en el cálculo del contorneado, el mismo se ejecuta una cantidad de veces mayor a la solicitada. Se modifica el algoritmo para corregir el error.
Se realiza una tercer prueba con el primer archivo la cual resulta totalmente satisfactoria.
Por último se realiza una prueba en acrílico con el 2do archivo, como se observa en la figura 6.13. La prueba resulta completamente satisfactoria.

6.2.2. Pruebas con controlador Fanuc en Centro de Mecanizado HAAS VF3YT

De igual forma a lo realizado en la fresadora EMCO, se realizan pruebas de simulación y posteriormente mecanizado en el centro de mecanizado HAAS.

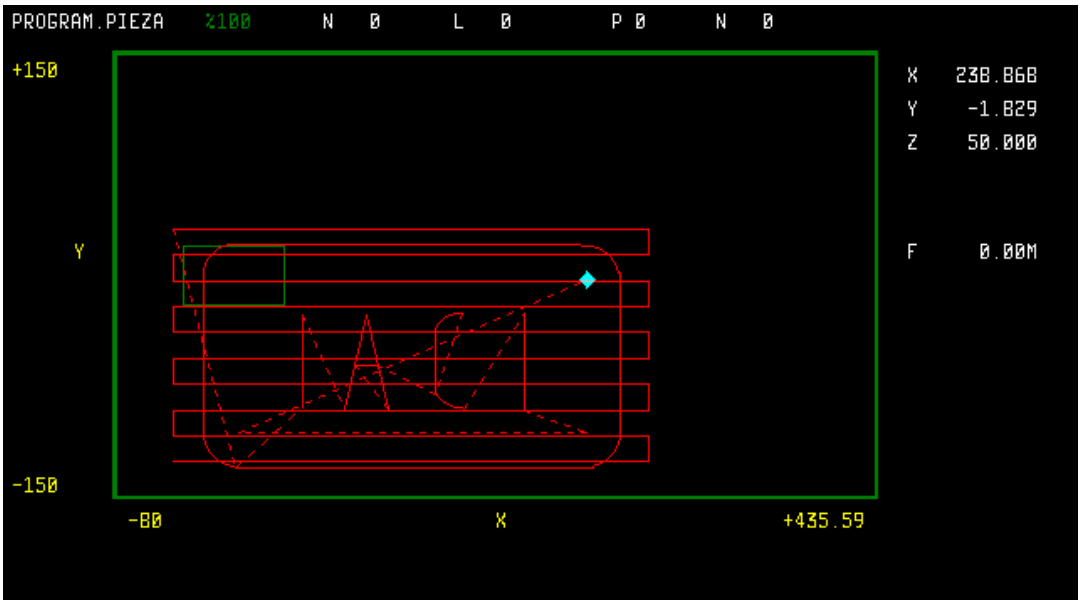


FIGURA 6.5: Resultado de la simulación del primer archivo

6.2.2.1. Pruebas de simulación

Se realizan pruebas de simulación de mecanizado con ambos archivos de prueba obteniendo resultados satisfactorios.
Los resultados de las pruebas se muestran en las figuras 6.14 y 6.15. “Panel de Dibujo”

6.2.2.2. Pruebas de mecanizado

Se realiza una prueba de mecanizado con el archivo de prueba 1.
Para esta prueba se testean los rasgos: contorneado, grabado y taladrado. El material utilizado es PVC gris rigido.
El resultado es satisfactorio, se puede observar el proceso de mecanizado y la pieza terminada en las figuras 6.16 y 6.17.

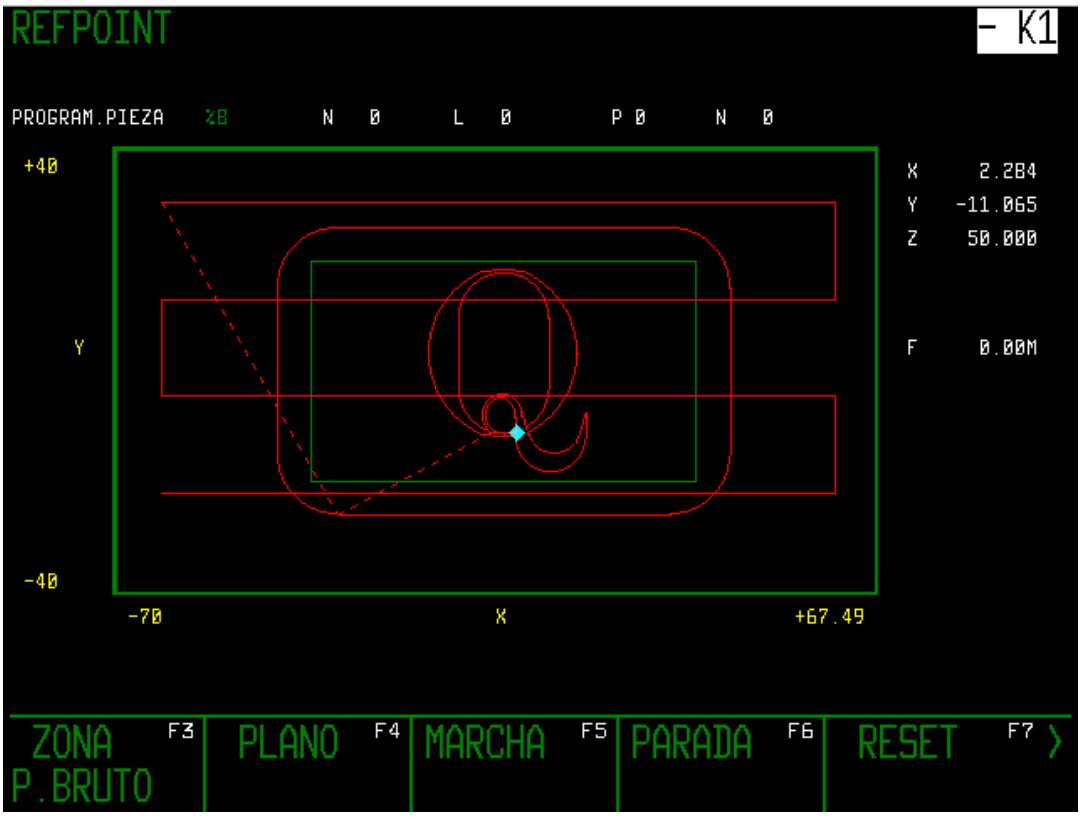


FIGURA 6.6: Resultado de la simulación del segundo archivo

Configuraciones | Herramientas | GCode

Herramienta	Planeado	Contornea...	Grabado	Taladrado
Nro	5	2	4	3
Diametro	40	6	1,5	5
Velocidad	1.000	2.000	2.000	1.200
Avance	400	600	600	70
Pasada	0,2	0,5	0,5	3
Z seguro	5	5	5	5
Z cambio	50	50	50	50

FIGURA 6.7: Detalle de herramientas utilizadas en la prueba

FIGURA 6.8: Ejemplo de configuración de 1 herramienta en WinNC

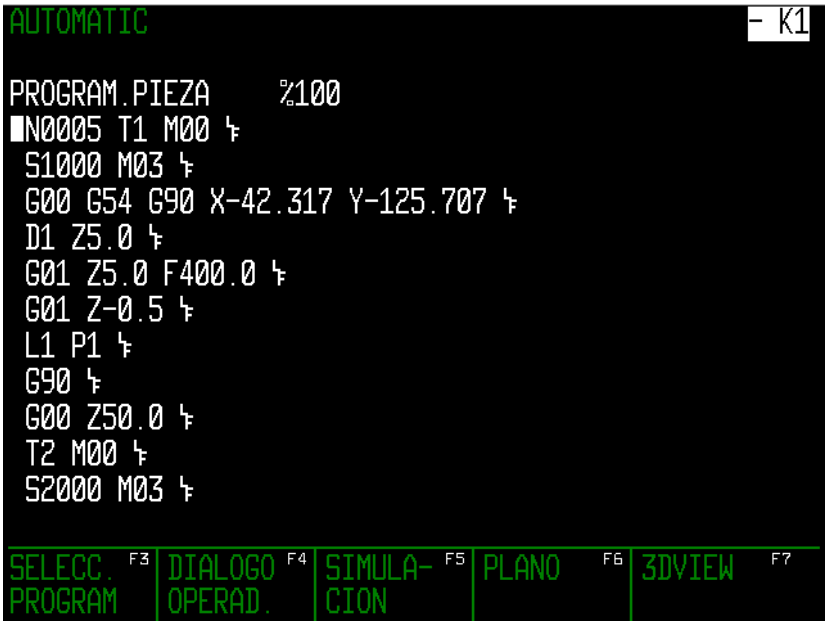


FIGURA 6.9: Visualización de un programa en WinNC

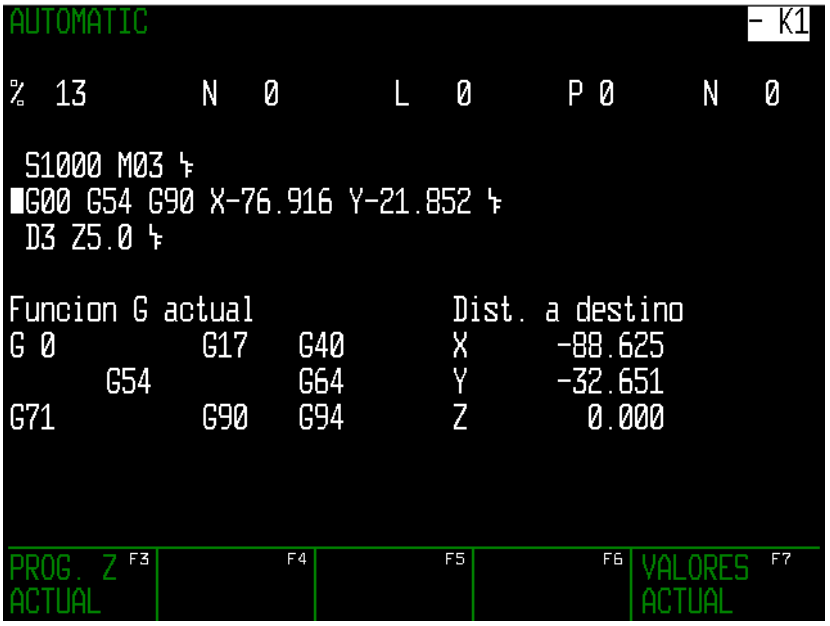


FIGURA 6.10: Proceso de mecanizado en marcha mostrando la secuencia actual

FIGURA 6.11: proceso de mecanizado en fresadora EMCO

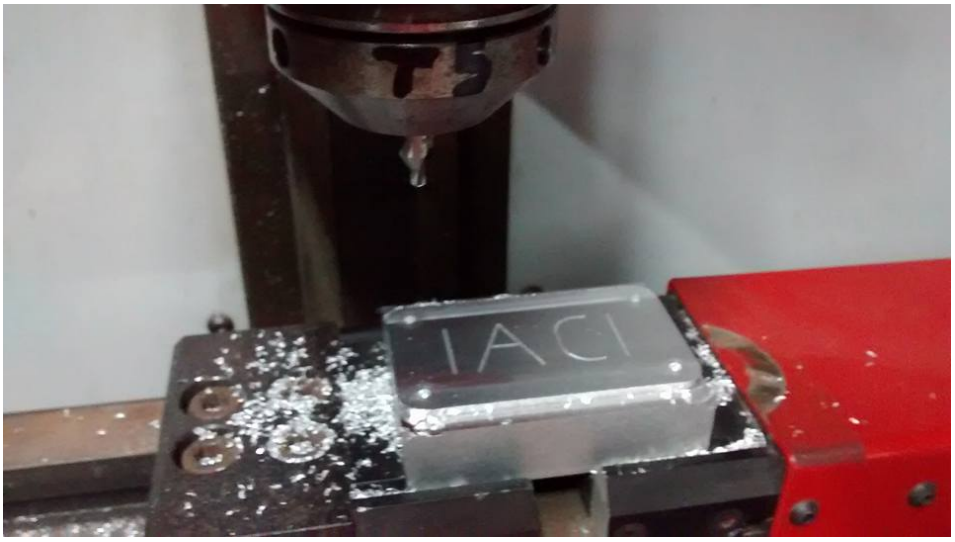


FIGURA 6.12: Pieza terminada con fresadora EMCO

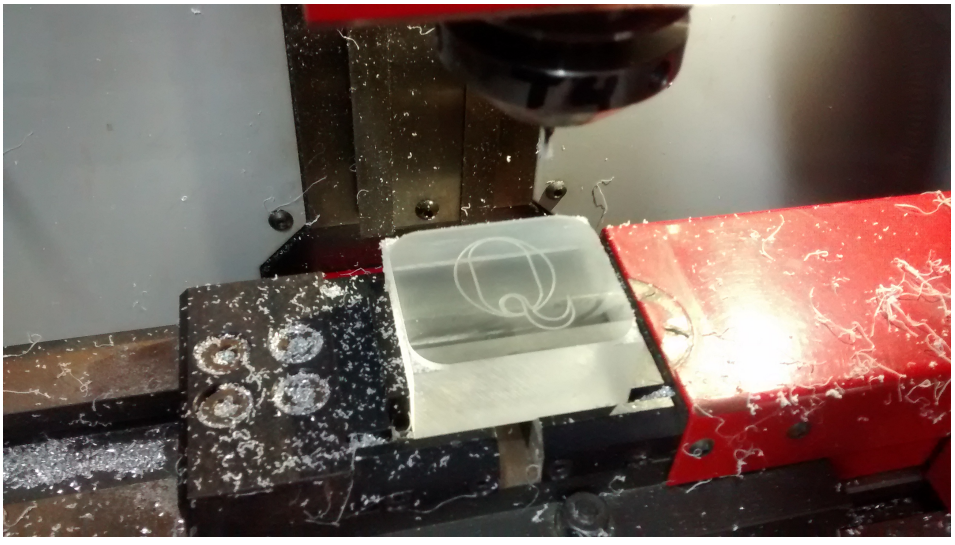


FIGURA 6.13: Pieza en acrílico terminada con fresadora EMCO

“Panel de Dibujo”

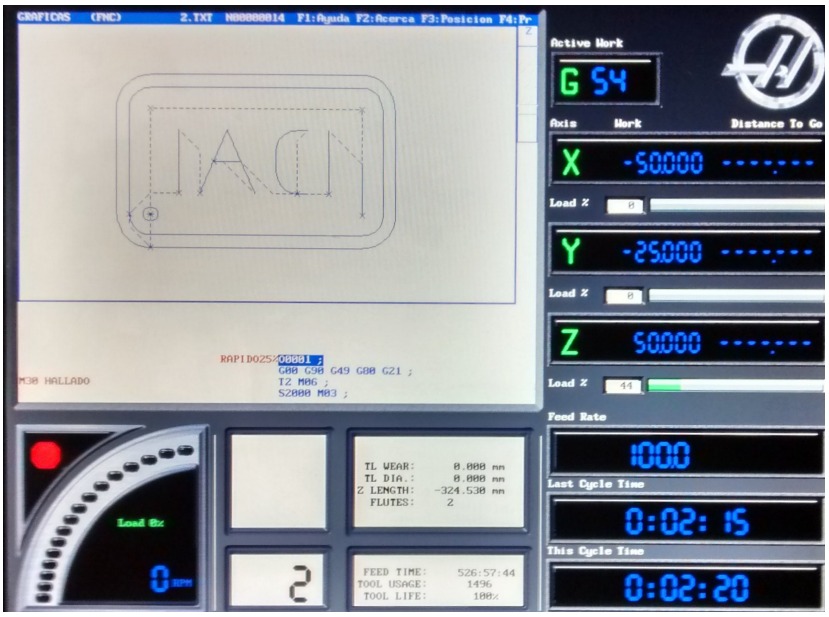


FIGURA 6.14: Prueba de simulación en Centro de Mecanizado HAAS

FIGURA 6.15: Prueba de simulación en Centro de Mecanizado HAAS

FIGURA 6.16: Prueba de mecanizado en marcha en Centro de Mecanizado HAAS

FIGURA 6.17: Resultado de la prueba de mecanizado en Centro de Mecanizado HAAS

Capítulo 7

Conclusiones

A lo largo de este proyecto fue necesario adquirir una gran cantidad de conocimientos, que luego se fueron aplicando, para así lograr los objetivos planteados.

7.1. Fresadora EMCO PC MILL 55

Se hizo un relevamiento de la fresadora EMCO PC MILL 55 de la Universidad Nacional de Quilmes, ya que la misma estaba en desuso desde hacía varios años.
Se corrigieron errores de inicio en la PC que controla a la fresadora y se pudo comprobar mediante distintas pruebas que la máquina en conjunto se encuentra en perfecto estado de funcionamiento.
Se hizo una investigación sobre como adaptar la PC que controla a la fresadora a las tecnologías modernas, ya que la misma está compuesta por hardware que actualmente se encuentra obsoleto.
Se plantearon tres posibles soluciones:

- Cambiar dicha PC por una PC actual: Para ello se hacía necesario contar con los discos de instalación y las licencias del software original de la fresadora, así como también adaptar las placas de comunicación entre la PC y la fresadora. La ventaja de esta opción era poder utilizar la misma PC para controlar el sistema y correr el software CAM, que se desarrollaría en un lenguaje de programación capaz de correr en una PC con tecnologías actuales.
- Instalar una placa USB en dicha PC para lograr el transporte de archivos entre distintas PC y de ésta forma no tener limitaciones a la hora de desarrollar el software CAM.
- Instalar una placa Ethernet en dicha PC con la misma metodología descrita en el punto anterior.

Se descartó la primer opción por ser de gran dificultad y con muchas posibilidades de errores, peligrando la finalización del proyecto en tiempo y forma.

La segunda posibilidad también fue descartada ya que la PC que controla la fresadora cuenta con Windows 95 como sistema operativo y dicho sistema no soporta unidades externas de almacenamiento tales como memorias flash.
Se implementó la tercer opción y se montó una red de área local entre la PC que controla a la fresadora y otra PC en la que se desarrollaría el software CAM. La primer red que se montó involucraba equipos con sistema operativo Windows. En una segunda etapa se realizó una red de área local entre la PC que controla la fresadora (Windows 95) y una PC con sistema operativo Ubuntu. Si bien este no era uno de los objetivos planteados, se consideró importante para probar la característica multiplataforma del software CAM que se desarrollaría.

7.2. Desarrollo de software:DXF2Machine

Paralelamente al relevamiento de la fresadora y la etapa de adaptación de la comunicación, se realizó una investigación sobre la forma de construcción de los archivos DXF, software CAD-CAM y lenguajes de programación.
Se optó por utilizar como lenguaje JAVA y como plataforma (o entorno de desarrollo) Eclipse, dando lugar en un futuro a una versión del software adaptada para Android.
De las distintas opciones de software CAD-CAM que se encuentran en el mercado, se tomaron las características mas relevantes.
De los desarrollos de código abierto obtenidos por internet, se decidió utilizar el proyecto DXF, desarrollado en EPSI París, que tiene como objetivo el manejo y edición de archivos DXF.
Se realizó un proceso de ingeniería inversa sobre el código del proyecto DXF. De esta forma, además de entender el funcionamiento del proyecto, se fue aprendiendo el lenguaje de programación JAVA y se logró una familiarización con el IDE Ecplise.
La información obtenida sirvió para identificar el punto de partida del software a realizar.
Se utilizaron todas las características de apertura y recolección de información de archivos DXF y se quitaron la mayoría de las herramientas de edición de entidades, ya que se planteó desarrollar un software CAM y no un CAD-CAM.
Se planteó incluir los rasgos básicos mecanizables de cualquier software CAM propietario, por lo tanto se investigaron y desarrollaron algoritmos para seleccionar y configurar los rasgos: planeado, grabado, taladrado y contorneado.
Si bien se planteó desarrollar un software específico para facilitar la tarea de programación y fomentar el uso de la fresadora EMCO PC MILL 55 de la Universidad Nacional de Quilmes, al plantear el proceso de desarrollo se observó que podía generarse un software adaptable a otras máquinas de una forma muy sencilla.
Se desarrollaron los algoritmos de traducción de los rasgos planteados y se adaptó la interfaz heredada del proyecto DXF para integrarla al proyecto DXF2Machine.
Finalmente se realizaron multiples pruebas con DXF2Machine en la fresadora EMCO PC MILL 55

de la Universidad Nacional de Quilmes y en un centro de mecanizado HAAS VF3YT que cedió para las pruebas la empresa metalúrgica EMT.

7.3. Resultado del proyecto planteado

De todo lo anterior se puede concluir que:

- Se cumplieron los objetivos planteados ampliamente.
- Se logró dejar en funcionamiento un equipo en desuso de la Universidad Nacional de Quilmes, como fuera la fresadora EMCO PC MILL 55.
- Se logró recuperar un equipo obsoleto e integrarlo con tecnologías actuales, tal es el caso de la PC controladora de la fresadora.
- Se logró desarrollar la base de un software CAM gratuito, multiplataforma y de código abierto.
- Se pudo probar el funcionamiento del software desarrollado tanto en un ámbito educativo como en la industria.

Capítulo 8

Mejoras a futuro

8.1. Comunicación

8.1.1. Red Wirless

Actualmente la comunicación está limitada a una red cableada. Con la tecnología actual podría investigarse el uso de una placa arduino conectada a la placa ethernet de la PC que controla la fresadora, para lograr configurar una red Wirless.

8.1.2. Actualización de Hardware

Podría evaluarse la idea del cambio de PC por una con tecnología actual, teniendo en cuenta las adaptaciones que eso conlleva.

8.2. DXF2Machine

La primer versión del software, si bien permite realizar una gran cantidad de mecanizados de distintas figuras, es una versión básica y en consecuencia presenta ciertas limitaciones. Algunas posibles mejoras que se podrían realizar:

8.2.1. Estudio de algoritmos de compensación y generación de trayectorias

Al momento de finalizar el presente proyecto, se desarrollo un algoritmo de compensación que funciona con ciertas combinaciones específicas de geometrías.

Los algoritmos de compensación son de gran interés para la industria e indispensables para lograr la generación de otros rasgos mecanizables tales como los vaciados.

8.2.2. Ampliación de entidades admitidas por el software

En la actualidad las entidades admitidas se limitan a Líneas, Arcos y Circunferencias. Si bien se puede realizar un gran número de piezas utilizando estas 3 entidades, sería conveniente incorporar: polilíneas, polígonos, texto, entre los mas destacados.

8.2.3. Ampliación de rasgos mecanizables

Ademas de planeado, contorneado, grabado y taladrado, en la industria se utilizan otros rasgos muy útiles como el mecanizado de cavidades (también conocido como vaciado o cajera), rebajes, contornos abiertos, roscados, etc.

8.2.4. Migración de JAVA a Android

Una característica de utilizar el IDE Eclipse como plataforma de programación, es la posibilidad de migrar el código de JAVA a sistemas Android. Si bien no es algo tribial, puede realizarse ampliando de esa forma la cantidad de sistemas en la que podría utilizarse DXF2Machine.

8.2.5. Ampliación de máquinas incluidas en el software para la traducción

Si bien DXF2Machine comenzó como un software para traducir archivos DXF a Código G, su desarrollo quedó planteado para lograr traducir dichos archivos al código de cualquier máquina (no necesariamente máquina herramienta). Para lograr esto basta con: conocer las características de lenguaje que interpreta la máquina que se desea agregar y, escribir un archivo ".properties" con los datos necesarios, siguiendo la estructura de las máquinas preexistentes.

Apéndice A

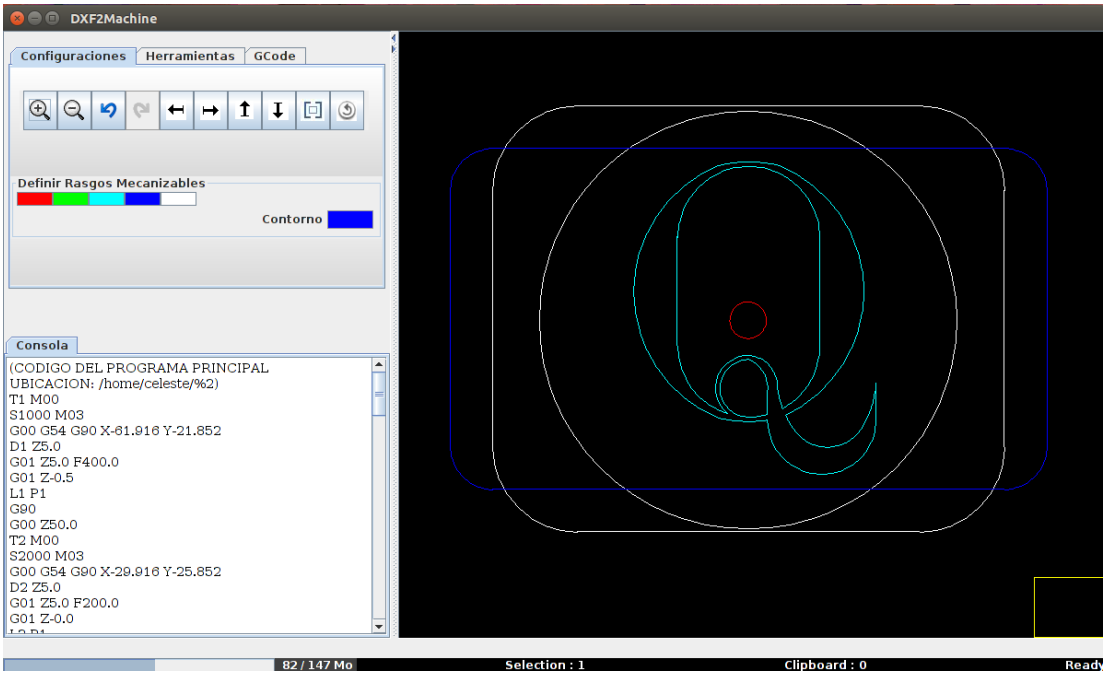
DXF2Machine: Manual del Usuario

Universidad Nacional de Quilmes

DXF2Machine: Manual del usuario

Manual del Usuario

Autor:Celeste Guagliano



Departamento de Ciencia y Tecnología

11 de julio de 2015

Índice general

Índice de figuras

Capítulo 1

Introduccion

En el presente manual se detallará cómo utilizar el software **DXFMachine** para abrir archivos DXF, configurar y generar mecanizados para las distintas maquinas compatibles con la versión 1.0 de dicho software.

1.1. Requisitos del sistema

Se requiere una PC con Java Runtime Enviroment 1.7 o superior.

Capítulo 2

Sobre la version 1.0

DXF2Machine es un software creado con el propósito de traducir archivos de extensión DXF (Drawing Exchange Format) a código G.
La versión 1.0 admite todas las versiones de archivos DXF-ASCII.
Actualmente solo admite 3 tipos de entidades: líneas, arcos y circunferencias. No son compatibles las demás entidades que pueden generarse en un archivo DXF.
Las entidades admitidas son la base de casi cualquier geometría. En la mayoría de los casos el usuario puede descomponer las entidades de su archivo en su editor habitual CAD, para obtener un archivo constituido por las entidades admitidas.
Si bien actualmente se incluyen solo dos postprocesadores con el software (Siemens Sinumerik y HAAS Fanuc), el mismo es de código abierto y fácilmente adaptable a cualquier otro controlador contando con su manual de programación.
Para los contorneados se admite cualquier combinación de lineas. En el caso de arcos y lineas, solo se admite intersecciones de arcos con lineas verticales u horizontales. La intersección entre arcos no está admitida en los contornos de la presente versión.

Capítulo 3

Descripcion del software

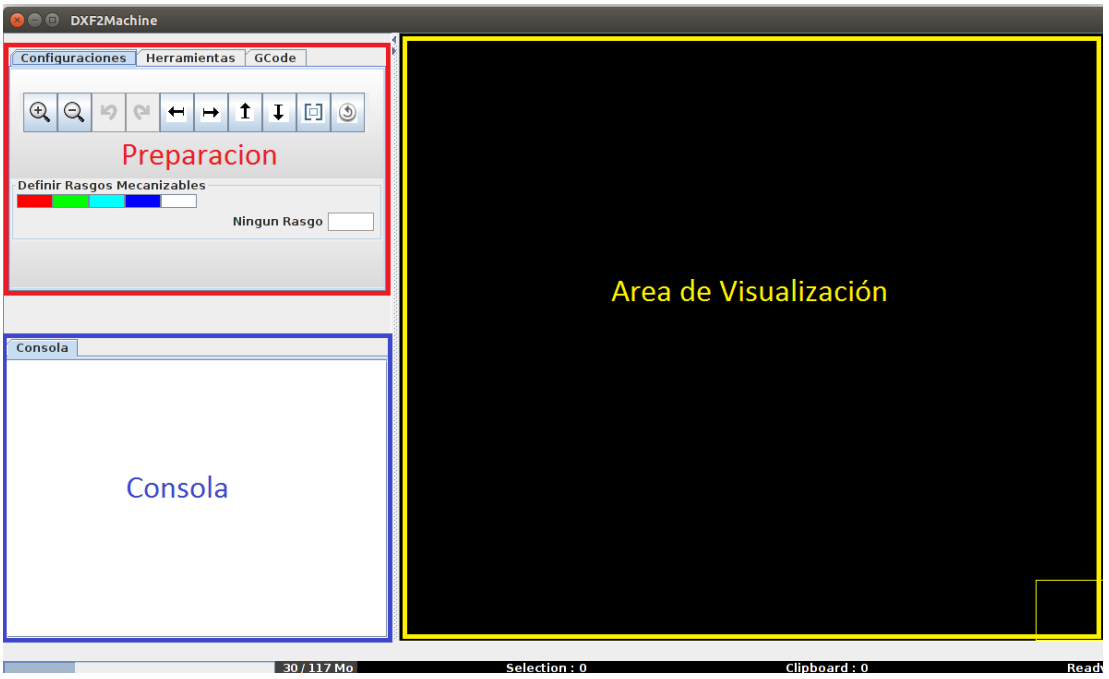


FIGURA 3.1: Descripción de la pantalla de DXF2Machine

Como se puede ver en la figura 3.1, la pantalla está divida en 3 regiones principales:

- Preparación: En esta región se encuentran las herramientas necesarias para configurar todo lo necesario para llevar a cabo el mecanizado deseado.
- Consola: En esta región se muestra el código resultante de la traducción. El mismo aparece coloreado según el mecanizado al cual corresponde y además presenta algunas aclaraciones útiles para el usuario.
- Area de visualización: En esta región se visualizan y se interactua con los planos de las geometrias a mecanizar.

Capítulo 4

Aprendiendo a utilizar DXF2Machine

Se describe mediante un ejemplo el uso del software.

4.1. Selección y configuración del plano

Para comenzar se abre un archivo compatible mediante el menú Archivo, opción Abrir... como se muestra en la figura 4.1. Una vez abierto el plano seleccionado, puede

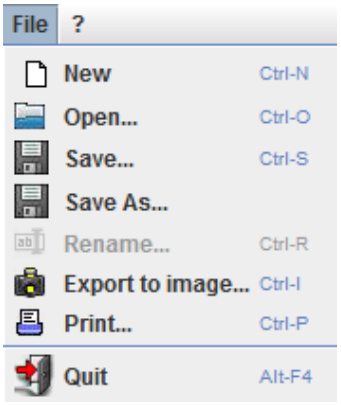


FIGURA 4.1: Menu Archivo

visualizarse en la zona de visualización. En la figura 4.2 se muestra el plano seleccionado para el ejemplo. El siguiente paso es configurar las entidades de las que se requiere traducción.

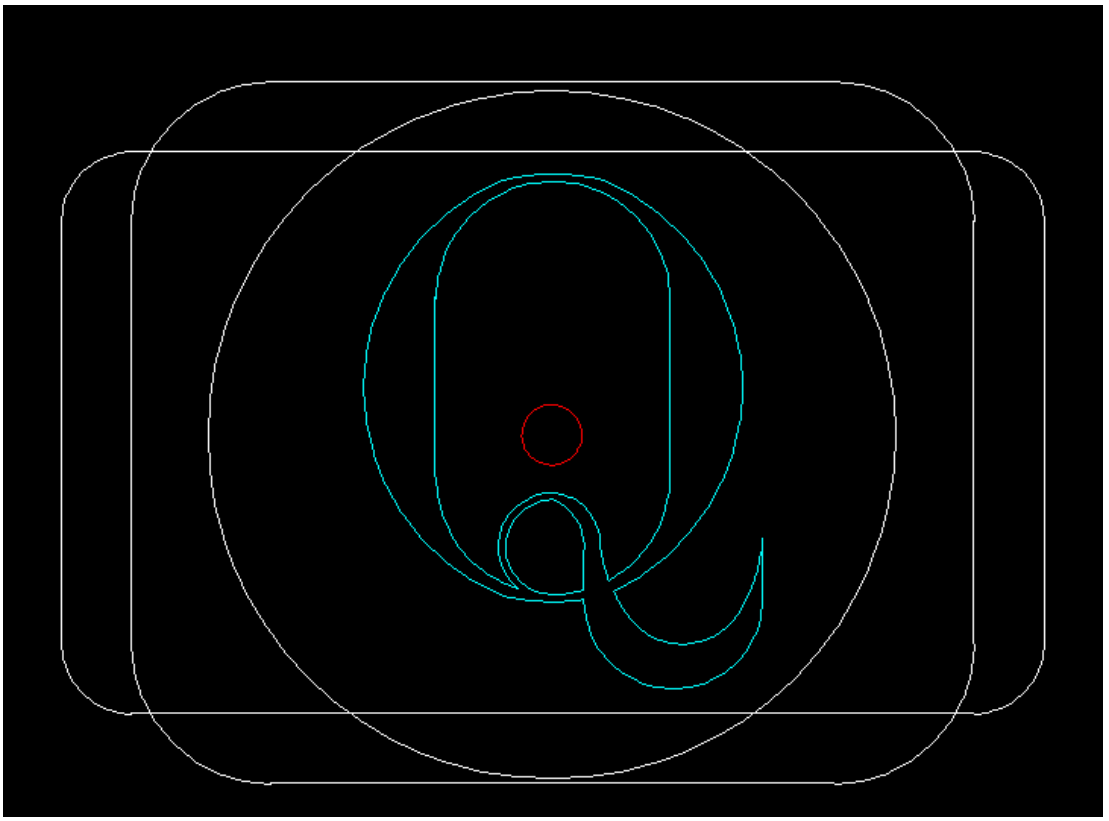


FIGURA 4.2: Visualización del plano seleccionado

4.2. Preparacion

Para lograr la obtención de la traducción deseada, se realizan 3 pasos preparatorios:

4.2.1. Configuraciones

Como se muestra en la figura 4.3, el software posee un código de colores para seleccionar las distintas entidades que componen los distintos mecanizados.

4.2.1.1. Origen

Para poder llevar a cabo el mecanizado, es imprescindible que el plano cuente con un origen.
Dicho origen coincide con el origen de la pieza que se configura en la máquina herramienta.
Para poder seleccionar el origen, el plano debe contar con una circunferencia ubicada en algún lugar. Es importante que la circunferencia no coincida con ningún rasgo mecanizable ya que no puede seleccionarse para ambas cosas.

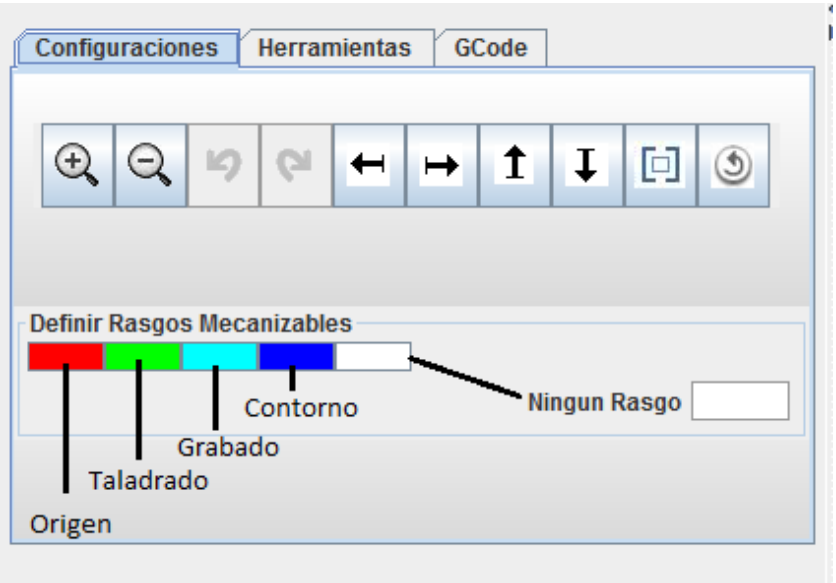


FIGURA 4.3: Solapa de Configuraciones

Para fijar el origen se debe realizar un click sobre el color rojo (Origen) en la solapa "Definir rasgos mecanizables" y luego seleccionar la circunferencia elegida.

4.2.1.2. Taladrado

Se conoce como taladrado a la operación de perforado. Las entidades admitidas en este rasgo son las circunferencias.
Seleccionando el color verde (taladrado), se pueden fijar todas las entidades a taladrar. Notar que el diámetro del taladrado está dado por el diámetro de la herramienta seleccionada y no por el diámetro de la circunferencia.

4.2.1.3. Grabado

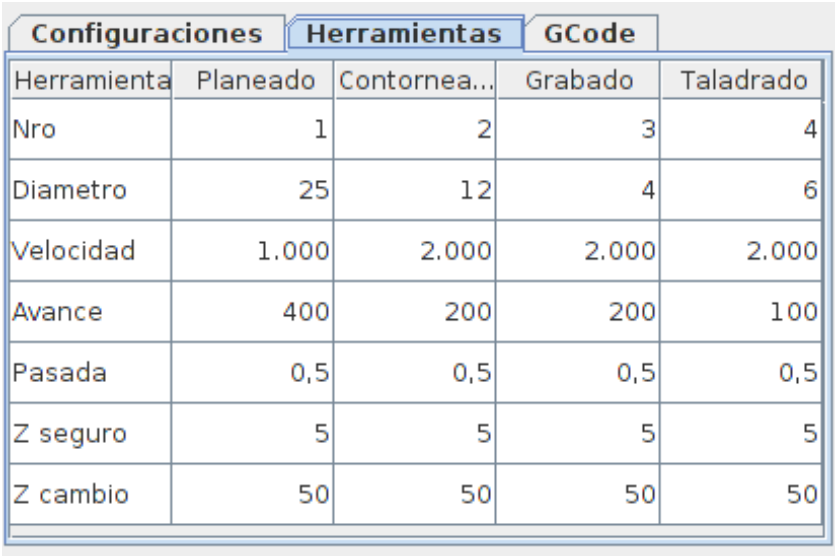
Se conoce como grabado a la operación en la que una geometría dada determina el camino del centro de la herramienta utilizada.
Seleccionando el color celeste (grabado), se pueden fijar todas las entidades que comprenden este mecanizado.
El grabado puede estar formado por cualquier combinación de las entidades admitidas.

4.2.1.4. Contorno

Se entiende por contorno toda aquella geometría cerrada compuesta por las entidades admitidas.

Seleccionando el color azul (contorno), podremos fijar un contorno del plano.
Esta selección se utilizará tanto para el tipo de mecanizado denominado contorneado, como para el planeado.
En la versión actual del software se admite un solo contorno cerrado por pieza.

4.2.2. Herramientas



Herramienta	Planeado	Contornea...	Grabado	Taladrado
Nro	1	2	3	4
Diametro	25	12	4	6
Velocidad	1.000	2.000	2.000	2.000
Avance	400	200	200	100
Pasada	0,5	0,5	0,5	0,5
Z seguro	5	5	5	5
Z cambio	50	50	50	50

FIGURA 4.4: Solapa de información de Herramientas

Se deben definir las características de la herramienta que se utilizan en cada mecanizado. Como se muestra en la figura 4.4, los datos a completar son:

- Nro= Es el número con el que se identifica la herramienta en la máquina para poder realizar los cambios y las compensaciones.
- Diámetro= Es el diámetro de corte de la herramienta.
- Velocidad (S)= Es la velocidad de giro del husillo, se calcula conociendo la velocidad de corte de la herramienta (provista por el fabricante de la misma). Para calcularla se utiliza la siguiente fórmula: $S = \frac{Vcx1000}{\pi xD}$, siendo D= diámetro de la herramienta.
- Avance (F)= Es el avance en mm/min de la herramienta. El mismo se calcula conociendo el dato de avance por corte provisto por el fabricante y utilizando la siguiente fórmula: $F = f_z xnxS$, siendo n la cantidad de cortes de la herramienta y S la velocidad de giro del husillo.
- Pasada= La pasada es la profundidad máxima a la que ingresa la herramienta cada vez que recorre la geometría.

- Zseguro= Es la altura por encima de la pieza a la que recurre la herramienta entre mecanizados.
- Zcambio= Es la altura a la que se debe desplazar la herramienta al finalizar un mecanizado para permitir el cambio a la siguiente (en el caso de que el cambio deba realizarse en forma manual).

4.2.3. GCode

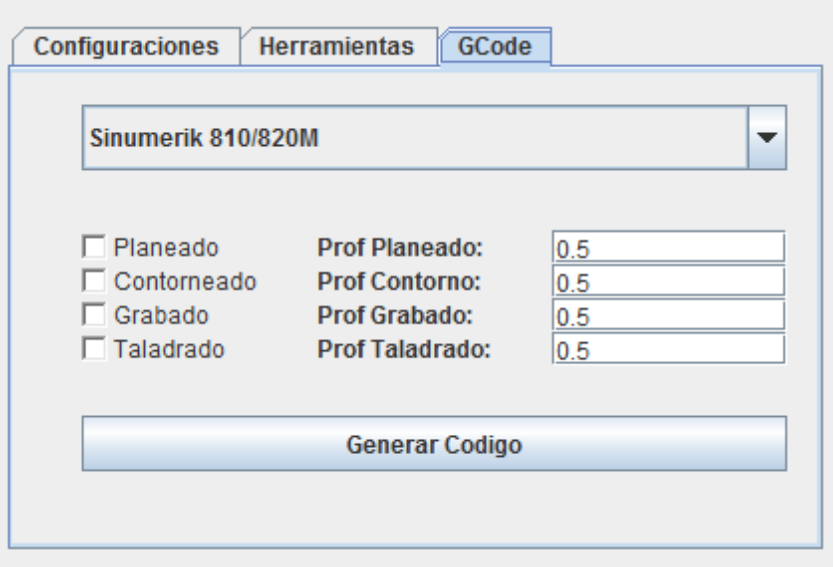


FIGURA 4.5: Solapa de selección de los rasgos a traducir

Teniendo los rasgos definidos y las herramientas configuradas, solo queda definir el postprocesador y los mecanizados a realizar.
Las opciones disponibles de postprocesadores se pueden ver en un menú desplegable, como se muestra en la figura 4.5.
Los mecanizados a realizar se pueden seleccionar mediante los checkbox correspondientes.
Entre los mecanizados disponibles se encuentran:

- Planeado: Calcula la trayectoria necesaria para cubrir toda la superficie encerrada por la geometría seleccionada como contorno, teniendo en cuenta el diámetro de la herramienta. La trayectoria calculada se repite tantas veces como sea necesario para completar la profundidad total teniendo en cuenta la pasada.
- Contorneado: Calcula la trayectoria necesaria para realizar el contorneado compensando el radio de la herramienta. Igual que en el caso anterior, la trayectoria se repetirá tantas veces como sea necesario para llegar a la profundidad total.

- Grabado: Sigue las geometrías designadas, al encontrar dos elementos que no tienen relación entre sí, mueve la herramienta hasta el plano de seguridad y luego se desplaza hasta la siguiente geometría para continuar el mecanizado. Repite estos movimientos hasta llegar a la profundidad total teniendo en cuenta la pasada.

- Taladrado: Selecciona los centros de las entidades a taladrar, realiza un chequeo para corroborar si la profundidad de pasada es menor a la profundidad total del agujero. Si es menor, se genera un ciclo de taladrado profundo, caso contrario se genera un ciclo normal de taladrado.

Con todos los pasos anteriores completados, se puede generar el código presionando el botón "Generar Codigo" que se muestra en la figura 4.5. Al realizar esta acción se despliega un cuadro de diálogo, el cual se muestra en la figura 4.6 en donde debe seleccionarse la carpeta de destino de los archivos generados.

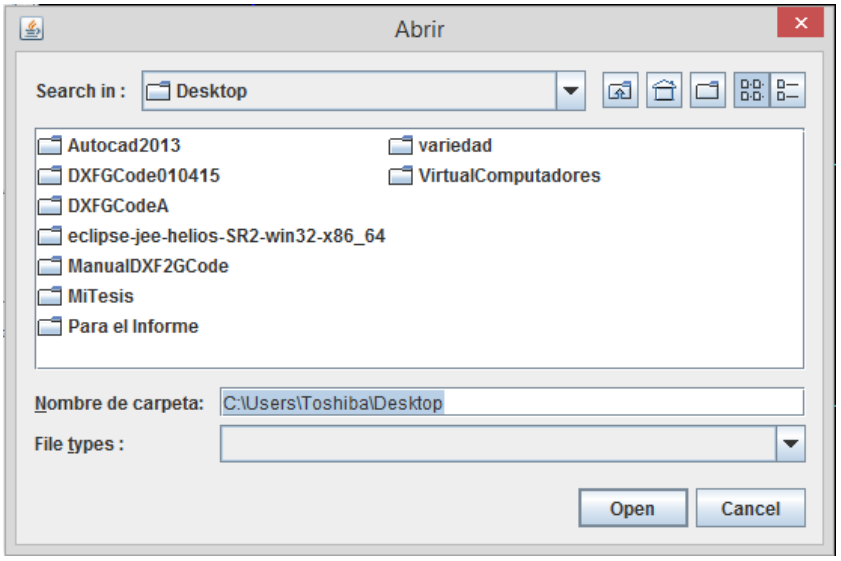


FIGURA 4.6: Cuadro de diálogo para selección de directorio destino

4.3. Visualizacion del codigo generado

Una vez seleccionado el directorio en donde se desea grabar el archivo, producto de la traducción, se puede ver en la consola el código generado.
En la figura 4.7 se muestra el código producido al traducir el rasgo grabado. Se puede ver que utilizando el mismo código de colores, se destacan los mecanizados generados para cada rasgo. Es importante notar que también se aclara cual es la ruta y el nombre del archivo generado.
Notar que en este caso se realiza el codigo para la fresadora EMCO PC Mill 55, por

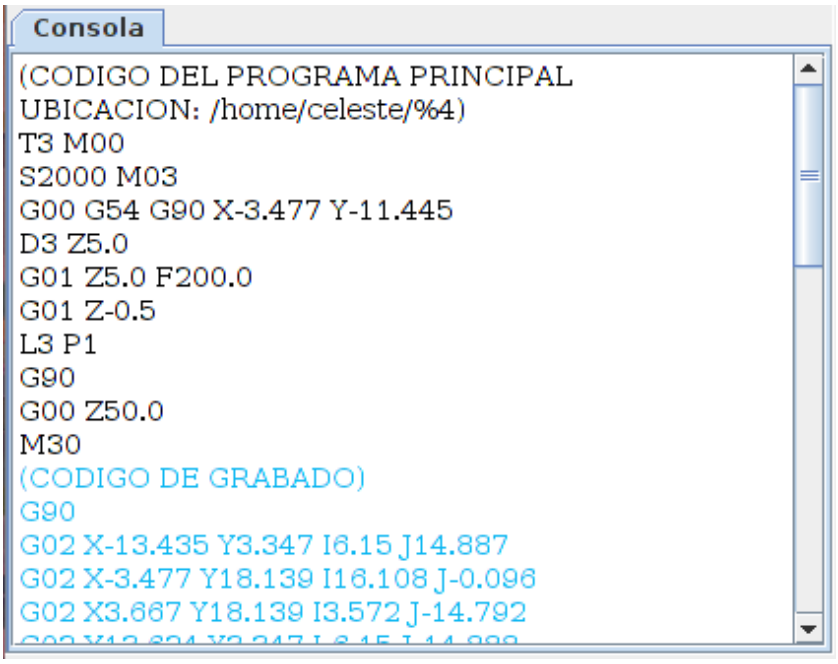


FIGURA 4.7: Visualización del código generado en la consola

lo que el código se encuentra en un programa principal de donde se hacen llamadas a subprogramas. Por este motivo el código del programa principal no incluye al código generado para el grabado.

Si en cambio el código hubiera sido generado para otra máquina, que utilice un único archivo con todo el código detallado, el código de grabado aparecería duplicado: formaría parte del programa principal y se lo identificaría con el mismo color de texto de ese archivo y al final apareceria explicitado en su color de configuración.

Apéndice B

DXF2Machine: Código del proyecto

```java
/*-------------------------------------------------------------------------------------
Copyright 2014, Celeste Gabriela Guagliano.

This file is part of DXF2Machine project.

DXF2Machine is free software: you can redistribute it and/or modify it under the terms of
the GNU General Public License as published by the Free Software Foundation, either
version 2 of the License.

DXF2Machine is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with DXF2Machine.
If not, see <http://www.gnu.org/licenses/>.
  -------------------------------------------------------------------------------------
  */

package cggColeccion;


import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.Enumeration;
import java.util.HashSet;
import java.util.Hashtable;
import java.util.Iterator;

import cggDatos.Coordenadas;
import cggDatos.DatosCirculo;
import cggDatos.DatosLinea;
import cggDatos.FormatoNumeros;
import cggDatos.Herramienta;
import cggDatos.datos;
import myDXF.DXF_Loader;

/**
 * This class defines util methods for colections.
 * @author: Celeste G. Guagliano
 * @version: 13/01/15
 *
 */


public class ColeccionFunciones {

    /** Method to select a sub-group of elements from a list.
     * @param listaEntidades is an entities hashtable.
     * @param condicion is the selection criteria.
     * @return a list of entities.
     */

    public static Hashtable ObtenerSubconjunto(HashSet listaEntidades,
            int condicion) {
        Hashtable aux = new Hashtable();
        for (Iterator it = listaEntidades.iterator(); it.hasNext();) {
            datos elemento = (datos) it.next();
            if (elemento.Color == condicion) {
                aux.put(aux.size() + 1, elemento);

            }
        }
```

```
62          return aux;
63      }
64
65      /** Method to get the initial coordinate of the first element in a list.
66       * @param lista is an ordered entities hashtable.
67       * @return the value of the initial coordinate of the first element of the list.
68       */
69
70      public static Coordenadas ObtenerCoordenadaInicial(Hashtable lista) {
71          datos inicial = (datos) lista.get(0);
72          double x1 = 0, y1 = 0;
73          Coordenadas coordInicial = new Coordenadas(0, 0);
74          if (inicial.orientacion == 0) {
75              coordInicial = new Coordenadas(inicial.ComienzoX, inicial.ComienzoY);
76          } else {
77              coordInicial = new Coordenadas(inicial.FinalX, inicial.FinalY);
78          }
79
80          return coordInicial;
81      }
82
83      /** Method to initialize an ordered list
84       * @param lista is an ordered entities hashtable.
85       * @return the key of the first ordered element.
86       */
87      public static int ObtenerElementoInicial(Hashtable lista) {
88          Hashtable iniciales = new Hashtable();
89          int llave = 0;
90          Coordenadas comparativa = new Coordenadas(0, 0);
91          comparativa = ObtenerCoordenadaMinima(lista);
92          iniciales = MinimaDistancia(lista, comparativa);
93          llave = SeleccionarUnInicial(iniciales, comparativa, lista);
94          return llave;
95      }
96
97      /** Method to decide between two possible first elements of an ordered list.
98       * @param iniciales is an ordered entities hashtable.
99       * @param comparativa it's the criteria for decision
100      * @param lista it's an ordered list of entities.
101      * @return the key of the selected element.
102      */
103     private static int SeleccionarUnInicial(Hashtable iniciales,
104             Coordenadas comparativa, Hashtable lista) {
105         // TODO Teniendo una lista de posibles entidades iniciales, este metodo aplica un
    criterio de selección para decidir cual es el inicial que se utiliza.
106         datos elemento = null;
107         if (iniciales.size() > 1) {
108             Coordenadas coinciden = new Coordenadas(0, 0);
109             coinciden = ObtenerCoincidente(iniciales);
110             double alfa1 = 0;
111             double alfa2 = 0;
112             datos elemento1 = (datos) iniciales.get(1);
113             datos elemento2 = (datos) iniciales.get(2);
114             double x1 = elemento1.ComienzoX;
115             double y1 = elemento1.ComienzoY;
116             double x2 = elemento2.ComienzoX;
117             double y2 = elemento2.ComienzoY;
118             if (x1 == coinciden.x && y1 == coinciden.y) {
119                 x1 = elemento1.FinalX;
120                 y1 = elemento1.FinalY;
121             } else {
122                 elemento1.orientacion = -1;
```

```
123             }
124             if (x2 == coinciden.x && y2 == coinciden.y) {
125                 x2 = elemento2.FinalX;
126                 y2 = elemento2.FinalY;
127             } else {
128                 elemento2.orientacion = -1;
129             }
130             double Xmin = comparativa.x;
131             double Ymin = comparativa.y;
132             alfa1 = Math.atan((y1 - Ymin) / (x1 - Xmin));
133             alfa2 = Math.atan((y2 - Ymin) / (x2 - Xmin));
134             if (alfa2 < alfa1) {
135                 elemento = elemento1;
136             } else {
137                 elemento = elemento2;
138             }
139 
140         } else {
141             elemento = (datos) iniciales.get(1);
142             elemento = ComprobarOrdenCoordenada(elemento, comparativa);
143         }
144         int clave = 0;
145         for (Enumeration k = lista.keys(); k.hasMoreElements();) {
146             int llave = (int) k.nextElement();
147             datos valor = (datos) lista.get(llave);
148             if (valor == elemento) {
149                 clave = llave;
150             }
151         }
152         return clave;
153     }
154 
155     /**
156      *  Method to define the orientation of an entity analyzing it's coordinates in the
    context of the list.
157      * @param elemento it's an entity.
158      * @param comparativa it's the evaluation criteria.
159      * @return an oriented element.
160      */
161 
162     private static datos ComprobarOrdenCoordenada(datos elemento,
163             Coordenadas comparativa) {
164         //TODO Este método decide si la entidad en cuestión se encuentra orientada en forma
    directa o inversa
165         // es decir, si sus coordenadas iniciales son las correctas en el conjunto o debe
    tomarse sus coordenadas finales como iniciales y viceversa.
166         double x1 = elemento.ComienzoX;
167         double y1 = elemento.ComienzoY;
168         double x2 = elemento.FinalX;
169         double y2 = elemento.FinalY;
170         double distancia1 = Math.sqrt(Math.pow(x1 - comparativa.x, 2)
171                 + Math.pow(y1 - comparativa.y, 2));
172         double distancia2 = Math.sqrt(Math.pow(x2 - comparativa.x, 2)
173                 + Math.pow(y2 - comparativa.y, 2));
174         if (distancia1 < distancia2) {
175             elemento.orientacion = 0;
176         } else {
177             elemento.orientacion = -1;
178         }
179         return elemento;
180     }
181 
```

```java
    /** Method to compare coordinates and get the matched pair.
     * @param iniciales it's a list of two entities.
     * @return the matched coordinate.
     */
    private static Coordenadas ObtenerCoincidente(Hashtable iniciales) {
        //TODO Este método compara las coordenas de dos entidades y obtiene un par coincidente.
        Coordenadas coincidentes = new Coordenadas(0, 0);
        datos elemento1 = (datos) iniciales.get(1);
        datos elemento2 = (datos) iniciales.get(2);
        double x1 = elemento1.ComienzoX;
        double y1 = elemento1.ComienzoY;
        double x2 = elemento2.ComienzoX;
        double y2 = elemento2.ComienzoY;
        if (x1 == x2 && y1 == y2) {
            coincidentes = new Coordenadas(x1, y1);
        } else {
            x1 = elemento1.FinalX;
            y1 = elemento1.FinalY;
            if (x1 == x2 && y1 == y2) {
                coincidentes = new Coordenadas(x1, y1);
            } else {
                x2 = elemento2.FinalX;
                y2 = elemento2.FinalY;
                if (x1 == x2 && y1 == y2) {
                    coincidentes = new Coordenadas(x1, y1);
                } else {
                    x1 = elemento1.ComienzoX;
                    y1 = elemento1.ComienzoY;
                    coincidentes = new Coordenadas(x1, y1);
                }
            }
        }
        return coincidentes;
    }

    /** Method to calculate the minimun distance between the elements of a list and a pair of comparative coordinates.
     * @param lista a list of entities.
     * @param comparativa the comparative coordinates.
     * @return a list of maximum two entities that meet the criteria.
     */
    private static Hashtable MinimaDistancia(Hashtable lista,
            Coordenadas comparativa) {
        //TODO Este método recibe una lista de entidades y calcula cual es la entidad que posee un par de coordenadas de distancia minima a un punto de referencia dado.
        Hashtable elementosIniciales = new Hashtable();
        datos inicial1 = null;
        for (Enumeration e = lista.keys(); e.hasMoreElements();) {
            int clave = (int) e.nextElement();
            inicial1 = (datos) lista.get(clave);
        }
        double distancia1 = CalcularDistanciAReferencia(comparativa, inicial1);
        int clave = 0;
        int clave2 = 0;
        datos inicial2 = null;
        double distancia2 = 0;
        for (Enumeration e = lista.keys(); e.hasMoreElements();) {
            int key = (int) e.nextElement();
            datos pinicial = (datos) lista.get(key);
            double distancia = CalcularDistanciAReferencia(comparativa,
                    pinicial);
```

```
241             if (distancia < distancia1) {
242                 clave = key;
243                 inicial1 = pinicial;
244                 distancia1 = distancia;
245             } else if (distancia == distancia1 && pinicial != inicial1) {
246                 clave2 = key;
247                 inicial2 = pinicial;
248             }
249         }
250         elementosIniciales.put(1, inicial1);
251         if (inicial2 != null) {
252             elementosIniciales.put(2, inicial2);
253         }
254 
255         return elementosIniciales;
256 
257     }
258 
259     /** Method to compare coordinates of entities
260      * @param inicial1 it's the data of an entity.
261      * @param inicial2 it's the data of an entity.
262      * @return true if there's a coincidence, false otherwise.
263      */
264         private static boolean compararEntidades(datos inicial1, datos inicial2) {
265         //TODO Este metodo compara entidades para saber si comparten un punto, es decir un
    par de coordenadas.
266         //mal puesto el nombre, debería ser "compararCoordenadasEntidades" o algo similar.
267         double x1 = inicial1.ComienzoX;
268         double x2 = inicial2.ComienzoX;
269         double y1 = inicial1.ComienzoY;
270         double y2 = inicial2.ComienzoY;
271         if (x1 == x2 && y1 == y2) {
272             return true;
273         } else {
274             x1 = inicial1.FinalX;
275             y1 = inicial1.FinalY;
276             if (x1 == x2 && y1 == y2) {
277                 return true;
278             } else {
279                 x2 = inicial2.FinalX;
280                 y2 = inicial2.FinalY;
281                 if (x1 == x2 && y1 == y2) {
282                     return true;
283                 } else {
284                     x1 = inicial1.ComienzoX;
285                     y1 = inicial1.ComienzoY;
286                     if (x1 == x2 && y1 == y2) {
287                         return true;
288                     } else {
289                         // TODO Auto-generated method stub
290                         return false;
291                     }
292                 }
293             }
294         }
295     }
296 
297         /** Method to calculate the distance between a couple of pair coordinates.
298          * @param comparativa it's the reference pair coordinate.
299          * @param inicial1 it's the element whose coordinates are analyzed
300          * @return the calculated distance.
301          */
```

```
302         private static double CalcularDistanciaAReferencia(Coordenadas comparativa,
303             datos inicial1) {
304         //TODO Este método calcula la distancia entre un par de coordenadas y un punto de
    referencia,
305         //compara el valor obtenido calculando con la coordenada inicial y final de la
    entidad y devuelve el menor.
306         double x1 = inicial1.ComienzoX;
307         double y1 = inicial1.ComienzoY;
308         double x2 = inicial1.FinalX;
309         double y2 = inicial1.FinalY;
310         double distancia1 = Math.sqrt(Math.pow(x1 - comparativa.x, 2)
311                 + Math.pow(y1 - comparativa.y, 2));
312         double distancia2 = Math.sqrt(Math.pow(x2 - comparativa.x, 2)
313                 + Math.pow(y2 - comparativa.y, 2));
314         if (distancia1 < distancia2) {
315             return distancia1;
316         } else {
317             return distancia2;
318         }
319 
320     }
321 
322         /** Method to get the minimum value of the x and y coordinates regardless of
    whether these values are from the same entity or not
323          * @param lista it's a list of entities.
324          * @return the minimun coordinates.
325          */
326         private static Coordenadas ObtenerCoordenadaMinima(Hashtable lista) {
327         //TODO recibe una lista y busca entre las coordenadas de sus entidades el menor
    valor de X y de Y, le resta un valor de seguridad y devuelve este par de coordenadas.
328         Coordenadas minimas = new Coordenadas(0, 0);
329         minimas = InicializarCoordenadas(lista);
330         double x = minimas.x;
331         double y = minimas.y;
332         for (Enumeration e = lista.elements(); e.hasMoreElements();) {
333             datos elemento = (datos) e.nextElement();
334             double nuevoX = elemento.ComienzoX;
335             double nuevoY = elemento.ComienzoY;
336             x = ChequearCoordenada(x, nuevoX);
337             y = ChequearCoordenada(y, nuevoY);
338             nuevoX = elemento.FinalX;
339             nuevoY = elemento.FinalY;
340             x = ChequearCoordenada(x, nuevoX);
341             y = ChequearCoordenada(y, nuevoY);
342         }
343         minimas = new Coordenadas(x - 2, y - 2);
344 
345         return minimas;
346     }
347 
348         /** Method to get a random pair of coordinates from a list
349          * @param lista it's a list of entities.
350          * @return a random pair of coordinates.
351          */
352 
353     private static Coordenadas InicializarCoordenadas(Hashtable lista) {
354         //TODO Este metodo toma un elemento cualquiera de una tabla y devuelve sus
    coordenadas iniciales.
355         datos elemento = new datos(0, 0, 0, 0, 0, false, 0, 0);
356         Coordenadas iniciales = new Coordenadas(0, 0);
357         for (Enumeration e = lista.elements(); e.hasMoreElements();) {
358             elemento = (datos) e.nextElement();
```

```
359         }
360         iniciales = new Coordenadas(elemento.ComienzoX, elemento.ComienzoY);
361         return iniciales;
362     }
363 
364     /** Method to compare two numbers.
365      * @param valor it's a number.
366      * @param nuevoValor it's a number.
367      * @return the minimun number.
368      */
369     private static double ChequearCoordenada(double valor, double nuevoValor) {
370         //TODO compara el valor de dos coordenadas y devuelve el menor.
371         if (nuevoValor < valor) {
372             return nuevoValor;
373         } else {
374             return valor;
375         }
376     }
377 
378     /** Method for generating an ordered list.
379      * @param ordenados it's a list of ordered entities.
380      * @param lista it's a list of entities to be ordered.
381      * @return an ordered list.
382      */
383     public static Hashtable ObtenerElementosOrdenados(Hashtable ordenados,
384             Hashtable lista) {
385         //TODO Toma las entidades de una tabla, las ordena y las coloca en una tabla
    siguiendo el orden.
386         //Devuelve la tabla ordenada.
387         ordenados = ObtenerSiguienteElemento(ordenados, lista);
388         return ordenados;
389     }
390 
391     /** Method to get the next element for an ordered list fulfilling certain criteria
392      * @param ordenados it's a list of ordered entities.
393      * @param lista it's a list of entities to be ordered.
394      * @return an ordered list.
395      */
396     private static Hashtable ObtenerSiguienteElemento(Hashtable ordenados,
397             Hashtable listaContorno) {
398         //TODO Este método toma el último valor de la tabla de elementos ordenados, busca
    el siguiente en la tabla no ordenada según el criterio de ordenamiento
399         // lo coloca en la tabla ordenada y lo saca de la tabla original.
400         do {
401             datos elemento = ComprobarCoordenadaInicial(ordenados,
402                     listaContorno);
403             if (elemento == null) {
404                 elemento = ComprobarCoordenadaFinal(ordenados, listaContorno);
405             }
406             if (elemento == null) {
407                 elemento = ObtenerElementoAleatorio(listaContorno);
408             }
409             int clave = ordenados.size() + 1;
410             ordenados.put(clave, elemento);
411             for (Enumeration e = listaContorno.keys(); e.hasMoreElements();) {
412                 int key = (int) e.nextElement();
413                 datos ele = (datos) listaContorno.get(key);
414                 if (ele == elemento) {
415                     listaContorno.remove(key);
416                 }
417             }
418         } while (listaContorno.size() > 1);
```

```
419         for (Enumeration e = listaContorno.keys(); e.hasMoreElements();) {
420             int key = (int) e.nextElement();
421             datos ele = (datos) listaContorno.get(key);
422             listaContorno.remove(key);
423             ordenados.put(ordenados.size() + 1, ele);
424         }
425         return ordenados;
426     }
427 
428     /** Method to get a random element of a list.
429      * @param lista it's a list of entities
430      * @return an element.
431      */
432     private static datos ObtenerElementoAleatorio(Hashtable listaContorno) {
433         //TODO devuelve un elemento al azar de la lista recibida.
434         int llave = 0;
435         llave = ObtenerElementoInicial(listaContorno);
436         datos elemento = (datos) listaContorno.get(llave);
437         return elemento;
438     }
439 
440     /** Method to compare if the final coordinate of the last item in an ordered list
    matches with the final coordinate of any other element of the orginal list.
441      * @param ordenados it's a list of ordered entities.
442      * @param lista it's a list of entities to be ordered.
443      * @return an element.
444      */
445     private static datos ComprobarCoordenadaFinal(Hashtable ordenados,
446             Hashtable lista) {
447         //TODO chequea si la coordenada final del ultimo elemento de una lista ordenada
    coincide con la coordenada final de algun otro elemento de la lista original.
448         datos elemento = null;
449         boolean elementoEncontrado = false;
450         Coordenadas FinalUltimoElementoOrdenado = ObtenerCoordenadaFinal(ordenados);
451         for (Enumeration e = lista.elements(); e.hasMoreElements();) {
452             datos siguiente = (datos) e.nextElement();
453             Coordenadas SiguienteElemento = new Coordenadas(siguiente.FinalX,
454                     siguiente.FinalY);
455             elementoEncontrado = CompararCoordenadas(
456                     FinalUltimoElementoOrdenado, SiguienteElemento);
457             if (elementoEncontrado == true) {
458                 elemento = siguiente;
459                 elemento.orientacion = -1;
460             }
461         }
462         return elemento;
463     }
464 
465     /** Method to compare a couple of pair coordinates.
466      * @param dato1 a pair coordinates.
467      * @param dato2 a pair coordinates
468      * @return true if the coordinates match, false otherwise.
469      */
470     private static boolean CompararCoordenadas(Coordenadas dato1,
471             Coordenadas dato2) {
472     //TODO comprueba si un par de coordenadas coinciden, si es así devuelve true, caso
    contrario devuelve false.
473         boolean coincidencia = false;
474         if (dato1.x == dato2.x && dato1.y == dato2.y) {
475             coincidencia = true;
476         }
477         return coincidencia;
```

```
478     }
479 
480     /** Method to compare if the final coordinate of the last item in an ordered list
    matches with the initial coordinate of any other element of the orginal list.
481      * @param ordenados it's a list of ordered entities.
482      * @param lista it's a list of entities to be ordered.
483      * @return an element.
484      */
485     private static datos ComprobarCoordenadaInicial(Hashtable ordenados,
486             Hashtable listaContorno) {
487         //TODO Chequea si la coordenada final del ultimo elemento de la tabla ordenada
    coincide con la coordenada inicial de algun elemento de la tabla original.
488         datos elemento = null;
489         boolean elementoEncontrado = false;
490         Coordenadas FinalUltimoElementoOrdenado = ObtenerCoordenadaFinal(ordenados);
491         for (Enumeration e = listaContorno.elements(); e.hasMoreElements();) {
492             datos siguiente = (datos) e.nextElement();
493             Coordenadas SiguienteElemento = new Coordenadas(
494                     siguiente.ComienzoX, siguiente.ComienzoY);
495             elementoEncontrado = CompararCoordenadas(
496                     FinalUltimoElementoOrdenado, SiguienteElemento);
497             if (elementoEncontrado == true) {
498                 elemento = siguiente;
499                 elemento.orientacion = 0;
500                 return elemento;
501             }
502         }
503 
504         return elemento;
505     }
506 
507 
508     /** Method to determine whether a list of elements form a closed contour.
509      * @param listaContornoOrdenada it's a list of ordered entities.
510      * @return true if the contour is closed, false otherwise.
511      */
512      public static boolean EvaluarContornoCerrado(Hashtable listaContornoOrdenada) {
513         // TODO Evalúa si hay continuidad entre los elementos de la tabla, es decir que
    todos los elementos se toquen entre si, incluyendo el ultimo con el primero.
514         Coordenadas iniciales = new Coordenadas(0, 0);
515         Coordenadas finales = new Coordenadas(0, 0);
516         int clave = 1;
517         boolean comparteCoordenada = true;
518         for (Enumeration k = listaContornoOrdenada.keys(); k.hasMoreElements();) {
519             clave = (int) k.nextElement();
520             datos elemento1 = (datos) listaContornoOrdenada.get(clave);
521             if (k.hasMoreElements()) {
522                 clave = (int) k.nextElement();
523                 datos elemento2 = (datos) listaContornoOrdenada.get(clave);
524                 if (comparteCoordenada == true) {
525                     comparteCoordenada = compararEntidades(elemento1, elemento2);
526                 }
527             }
528         }
529         if (comparteCoordenada == true) {
530             datos elemento1 = (datos) listaContornoOrdenada.get(1);
531             datos elemento2 = (datos) listaContornoOrdenada
532                     .get(listaContornoOrdenada.size());
533             comparteCoordenada = compararEntidades(elemento1, elemento2);
534         }
535         return comparteCoordenada;
536     }
```

```
537
538     /** Method to orientate a closed contour.
539      * @param listaOrdenada it's a list of ordered entities.
540      * @return a list of orientated elements.
541      */
542
543     public static Hashtable OrientarContornoCerrado(Hashtable listaOrdenada) {
544         //TODO revisa la orientación de los elementos ordenados en la tabla y la corrije si
    es necesario.
545         int clave = 1;
546         if(listaOrdenada.size()>1){
547         do {
548             datos elemento1 = (datos) listaOrdenada.get(clave);
549             clave += 1;
550             datos elemento2 = (datos) listaOrdenada.get(clave);
551             listaOrdenada.remove(clave);
552             elemento2 = OrientarElemento(elemento1, elemento2);
553             listaOrdenada.put(clave, elemento2);
554           } while (clave < listaOrdenada.size());
555         }
556         return listaOrdenada;
557     }
558
559 /*  private static datos OrientarElementoFinal(datos elemento1, datos elemento2) {
560         // TODO Auto-generated method stub
561         return null;
562     }*/
563
564     /** Method to orientate an element using another element as a criterion.
565      * @param elemento1 the element to be oriented
566      * @param elemento2 the criteria of orientation.
567      * @return an oriented element.
568      */
569     private static datos OrientarElemento(datos elemento1, datos elemento2) {
570         //TODO Chequea cuales son las coordenadas que se tocan entre dos elementos y en
    base a esto orienta los mismos.
571         double x1 = 0, y1 = 0, x2 = 0, y2 = 0;
572         if (elemento1.orientacion == 0) {
573             x1 = elemento1.FinalX;
574             y1 = elemento1.FinalY;
575         } else {
576             x1 = elemento1.ComienzoX;
577             y1 = elemento1.ComienzoY;
578         }
579         x2 = elemento2.ComienzoX;
580         y2 = elemento2.ComienzoY;
581         if (x1 == x2 && y2 == y1) {
582             elemento2.orientacion = 0;
583         } else {
584             elemento2.orientacion = -1;
585         }
586         return elemento2;
587     }
588
589
590     /** Method to orientate an element using another element as a criterion.
591      * @param elemento1 the element to be oriented
592      * @param elemento2 the criteria of orientation.
593      * @return an oriented element.
594      */
595     private static Coordenadas ObtenerCoordenadaFinal(
596             Hashtable listaContornoOrdenada) {
```

```
597         //TODO Devuelve la coordenada final de un elemento teniendo en cuenta su
    orientación.
598         Coordenadas coordFinales = new Coordenadas(0, 0);
599         datos elementoFinal = (datos) listaContornoOrdenada
600                 .get(listaContornoOrdenada.size());
601         if (elementoFinal.orientacion == 0) {
602             coordFinales = new Coordenadas(elementoFinal.FinalX,
603                     elementoFinal.FinalY);
604         } else {
605             coordFinales = new Coordenadas(elementoFinal.ComienzoX,
606                     elementoFinal.ComienzoY);
607         }
608 
609         return coordFinales;
610     }
611 
612     /** Method to select the entity in the original list that meets the criteria to
    initialize the ordered list.
613      * @param datos is an entities list.
614      * @return a list of ordered entities.
615      */
616     public static Hashtable InicializarTablaOrdenada(Hashtable datos) {
617         //TODO coloca el elemento inicial en la tabla ordenada.
618         Hashtable ordenada = new Hashtable();
619         int elementoInicial = ObtenerElementoInicial(datos);
620         datos elemento = (datos) datos.get(elementoInicial);
621         ordenada.put(1, elemento);
622         return ordenada;
623     }
624 
625     /** Method to check if two entities share any pair of coordinates.
626      * @param elemento1 is an entity.
627      * @param elemento2 is an entity.
628      * @return true if any coordinates match, false otherwise.
629      */
630     public static boolean compartenCoordenada(datos elemento1, datos elemento2) {
631         //TODO chequea si dos elementos comparten coordenadas.
632         boolean comparten = false;
633         double x1 = 0, y1 = 0, x2 = 0, y2 = 0;
634         if (elemento1.orientacion == 0) {
635             x1 = elemento1.FinalX;
636             y1 = elemento1.FinalY;
637         } else {
638             x1 = elemento1.ComienzoX;
639             y1 = elemento1.ComienzoY;
640         }
641         if (elemento2.orientacion == 0) {
642             x2 = elemento2.ComienzoX;
643             y2 = elemento2.ComienzoY;
644         } else {
645             x2 = elemento2.FinalX;
646             y2 = elemento2.FinalY;
647         }
648         if (x1 == x2 && y1 == y2) {
649             comparten = true;
650         }
651         return comparten;
652     }
653 
654     /** Method to remove unneeded elements of a list.
655      * @param lista is the original list
656      * @param listaOptimizada is the resultant list.
```

```
657      * @return a new list.
658      */
659     public static Hashtable ObtenerNuevaLista(Hashtable lista,
660             Hashtable listaOptimizada) {
661         //TODO elimina de la tabla original los elementos que ya fueron incluidos en la
    tabla ordenada.
662         for (Enumeration k = lista.keys(); k.hasMoreElements();) {
663             int clave = (int) k.nextElement();
664             datos elemento = (datos) lista.get(clave);
665             if (listaOptimizada.contains(elemento)) {
666                 lista.remove(clave);
667             }
668         }
669         return lista;
670     }
671     /** This method calculates the trajectory needed to cover a surface with a tool of a
    certain diameter, based on the outline.
672      * @param listaContorno is the outline
673      * @param herramientas is the list of tools where belongs the one involve into this
    operation.
674      * @return a new list.
675      */
676     public static Hashtable ObtenerTocho(Hashtable listaContorno,
677             Hashtable herramientas) {
678         // TODO Encuentra las coordenadas maximas y minimas del contorno, y teniendo en
    cuenta estas coordenadas y la herramienta a utilizar calcula la tabla de trayectoria del
    planeado
679         Hashtable tocho = new Hashtable();
680         Coordenadas inicial = ObtenerCoordenadaMinima(listaContorno);
681         Coordenadas finales = ObtenerCoordenadaMaxima(listaContorno);
682         DatosCirculo elemento = chequearElementos(listaContorno);
683         if (elemento != null) {
684             inicial.x =(double) FormatoNumeros.formatearNumero(elemento.CentroX -
    elemento.Radio);
685             inicial.y = (double) FormatoNumeros.formatearNumero(elemento.CentroY -
    elemento.Radio);
686             finales.x = (double) FormatoNumeros.formatearNumero(elemento.CentroX +
    elemento.Radio);
687             finales.y = (double) FormatoNumeros.formatearNumero(elemento.CentroY +
    elemento.Radio);
688         }
689         Herramienta planeado = (Herramienta) herramientas.get("Planeado");
690         double diametro = planeado.Diametro;
691         tocho = GenerarTablaTocho(inicial, finales, diametro);
692         return tocho;
693     }
694
695     /** This method checks if there is any circle in a list.
696      * @param listaContorno is the list
697      * @return a circle if there's any in the list, null otherwise.
698      */
699     private static DatosCirculo chequearElementos(Hashtable listaContorno) {
700         // TODO chequea si los elementos de la lista son circulos.
701         DatosCirculo elemento = null;
702         for (Enumeration e = listaContorno.elements(); e.hasMoreElements();) {
703             datos ele = (datos) e.nextElement();
704             if (ele instanceof DatosCirculo) {
705                 elemento = (DatosCirculo) ele;
706             }
707         }
708         return elemento;
709     }
```

```
710
711     /** This method calculates the trajectory needed to cover a surface between a start
    point and an end point, increasing 70% of the tool diameter from line to line.
712      * @param inicial is the beginning point
713      * @param final is the finishing point.
714      * @param diametro is the tool diameter.
715      * @return a list.
716      */
717     private static Hashtable GenerarTablaTocho(Coordenadas inicial,
718             Coordenadas finales, double diametro) {
719         //TODO genera la tabla de trayectorias del planeado
720         Hashtable tocho = new Hashtable();
721         inicial.x =(double) FormatoNumeros.formatearNumero(inicial.x    - diametro);
722         finales.x =(double) FormatoNumeros.formatearNumero(finales.x    + diametro);
723         DatosLinea primerLinea = new DatosLinea(inicial.x, inicial.y,
724                 finales.x, inicial.y, 0, false, 0, 0);
725         tocho.put(1, primerLinea);
726         DatosLinea linea = new DatosLinea(0, 0, 0, 0, 0, false, 0, 0);
727         DatosLinea anterior = new DatosLinea(0, 0, 0, 0, 0, false, 0, 0);
728         DatosLinea arriba = new DatosLinea(0, 0, 0, 0, 0, false, 0, 0);
729         int key = 1;
730         do {
731             anterior = (DatosLinea) tocho.get(key);
732             arriba = DesplazateVertical(anterior, diametro);
733             linea = DesplazateHorizontal(arriba, diametro, inicial, finales);
734             key += 1;
735             tocho.put(key, arriba);
736             key += 1;
737             tocho.put(key, linea);
738         } while (linea.FinalY < finales.y);
739         return tocho;
740     }
741
742
743     /** This method calculates an horizontal line from an initial point with a certain
    distance and orientation.
744      * @param arriba is the beginning point
745      * @param diametro is the tool diameter.
746      * @param inicial is a point.
747      * @param finales is a point.
748      * @return an entity.
749      */
750     private static DatosLinea DesplazateHorizontal(DatosLinea arriba,
751             double diametro, Coordenadas inicial, Coordenadas finales) {
752         //TODO genera una linea horizontal
753         DatosLinea horizontal = new DatosLinea(0, 0, 0, 0, 0, false, 0, 0);
754         horizontal.ComienzoX = arriba.FinalX;
755         horizontal.ComienzoY = arriba.FinalY;
756         horizontal.FinalY = arriba.FinalY;
757         if (arriba.FinalX == inicial.x) {
758             horizontal.FinalX = finales.x;
759         } else {
760             horizontal.FinalX = inicial.x;
761         }
762         return horizontal;
763     }
764
765     /** This method calculates a vertical line from an initial point with a certain
    distance and orientation.
766      * @param anterior is the beginning point
767      * @param diametro is the tool diameter.
768      * @return an entity.
```

```
769     */
770    private static DatosLinea DesplazateVertical(DatosLinea anterior,
771            double diametro) {
772        //TODO genera una linea vertical
773        DatosLinea arriba = new DatosLinea(0, 0, 0, 0, 0, false, 0, 0);
774        arriba.ComienzoX = anterior.FinalX;
775        arriba.FinalX = anterior.FinalX;
776        arriba.ComienzoY = anterior.FinalY;
777        arriba.FinalY = (double) FormatoNumeros.formatearNumero(anterior.FinalY+ .7 *
   diametro);
778        return arriba;
779    }
780
781    /** This method search for the maximum value of X and Y in the entities of a list.
782     * @param listaContorno is an entities list
783     * @return a coordinate.
784     */
785    private static Coordenadas ObtenerCoordenadaMaxima(Hashtable listaContorno) {
786        // TODO obtiene el maximo valor de X e Y entre las coordenadas de la lista recibida
787        Coordenadas maximas = new Coordenadas(0, 0);
788        maximas = InicializarCoordenadas(listaContorno);
789        double x = maximas.x;
790        double y = maximas.y;
791        for (Enumeration e = listaContorno.elements(); e.hasMoreElements();) {
792            datos elemento = (datos) e.nextElement();
793            double nuevoX = elemento.ComienzoX;
794            double nuevoY = elemento.ComienzoY;
795            x = ChequearCoordenadaMax(x, nuevoX);
796            y = ChequearCoordenadaMax(y, nuevoY);
797            nuevoX = elemento.FinalX;
798            nuevoY = elemento.FinalY;
799            x = ChequearCoordenadaMax(x, nuevoX);
800            y = ChequearCoordenadaMax(y, nuevoY);
801        }
802        maximas = new Coordenadas(x, y);
803
804        return maximas;
805    }
806
807    /** This method compares two numbers and decides wich one is the biggest.
808     * @param x is a number.
809     * @param nuevoX is a number.
810     * @return a number.
811     */
812    private static double ChequearCoordenadaMax(double x, double nuevoX) {
813        //TODO compara el par de dos coordenadas y devuelve el mayor
814        if (nuevoX > x) {
815            return nuevoX;
816        } else {
817            return x;
818            // TODO Auto-generated method stub
819        }
820
821    }
822
823    /** This method checks the orientation of an entity and returns the final point.
824     * @param inicial is the representation of an entity
825     * @return a point.
826     */
827    public static Coordenadas ObtenerCoordenadaFinEntidad(datos inicial) {
828        //TODO devuelve el valor de la coordenada final de una entidad teniendo en cuenta
   su orientacion
```

```
829         Coordenadas finales = new Coordenadas(0, 0);
830         if (inicial.orientacion == 0) {
831             finales.x = inicial.FinalX;
832             finales.y = inicial.FinalY;
833         } else {
834             finales.x = inicial.ComienzoX;
835             finales.y = inicial.ComienzoY;
836                     }
837         return finales;
838     }
839     /** This method checks the orientation of an entity and returns the beginning point.
840      * @param inicial is the representation of an entity
841      * @return a point.
842      */
843     public static Coordenadas ObtenerCoordenadaInicioEntidad(datos inicial) {
844         //TODO devuelve el valor de la coordenada inicial de una entidad teniendo en cuenta
    su orientacion
845         Coordenadas iniciales = new Coordenadas(0, 0);
846         if (inicial.orientacion == 0) {
847             iniciales.x = inicial.ComienzoX;
848             iniciales.y = inicial.ComienzoY;
849         } else {
850             iniciales.x = inicial.FinalX;
851             iniciales.y = inicial.FinalY;
852         }
853         return iniciales;
854     }
855
856     /** This method checks if an entity is a circle and returns it's center.
857      * @param datos is the representation of an entity
858      * @return a point.
859      */
860     public static Coordenadas ObtenerCoordenadaCentroEntidad(datos datos) {
861         //TODO Obtiene la coordenada del centro de una entidad siempre que se trate de un
    circulo.
862         Coordenadas centro= new Coordenadas(0,0);
863         if(datos instanceof DatosCirculo){
864             centro= new
    Coordenadas(((DatosCirculo)datos).CentroX,((DatosCirculo)datos).CentroY);
865         }
866         return centro;
867     }
868
869     /** This method collect the circles of a list.
870      * @param lista is a list of entities.
871      * @return a list of circles.
872      */
873     public static Hashtable ObtenerCirculos(Hashtable lista) {
874         // TODO extrae los circulos de una lista dada y los devuelve en otra lista.
875         Hashtable seleccion= new Hashtable();
876         for (Enumeration e = lista.elements(); e.hasMoreElements();){
877             datos elemento = (datos) e.nextElement();
878             if(elemento instanceof DatosCirculo){
879                 seleccion.put(seleccion.size()+1, elemento);
880             }
881
882         }
883             return seleccion;
884     }
885
886 }
```

```java
/*-------------------------------------------------------------------------------------
Copyright 2014, Celeste Gabriela Guagliano.

This file is part of DXF2Machine project.

DXF2Machine is free software: you can redistribute it and/or modify it under the terms of
the GNU General Public License as published by the Free Software Foundation, either
version 2 of the License.

DXF2Machine is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with DXF2Machine.
If not, see <http://www.gnu.org/licenses/>.
  -------------------------------------------------------------------------------------*/

package cggDatos;

/**
 * This class defines the structure of Coordenadas.
 * Coordenadas is a data structure that stores a pair of coordinates.
 * @author: Celeste G. Guagliano
 * @version: 13/01/15
 *
 */

public class Coordenadas {
    public double x = 0;
    public double y = 0;

    public Coordenadas(double x1, double y1) {
        this.x = x1;
        this.y = y1;
    }

}

```

```
/*-------------------------------------------------------------------------------------
Copyright 2014, Celeste Gabriela Guagliano.

This file is part of DXF2Machine project.

DXF2Machine is free software: you can redistribute it and/or modify it under the terms
of the GNU General Public License as published by the Free Software Foundation, either
version 2 of the License.

DXF2Machine is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with DXF2Machine.
If not, see <http://www.gnu.org/licenses/>.
-------------------------------------------------------------------------------------*/


package cggDatos;


import cggGCode.GCode;
/**
 * This class defines the structure of datos and it's methods.
 * datos is a data structure that store the basic data of any entity.
 * @author: Celeste G. Guagliano
 * @version: 13/01/15
 *
 */


public class datos {

    public double ComienzoX;
    public double ComienzoY;
    public double FinalX;
    public double FinalY;
    public int Color;
    public boolean ubicado = false;
    public int posicion = 0;
    public int orientacion = 0;


    public datos(double d, double e, double f, double g, int _color,
            boolean ubicado, int posicion, int orientacion) {

        this.ComienzoX = d;
        this.ComienzoY = e;
        this.FinalX = f;
        this.FinalY = g;
        this.Color = _color;
        this.ubicado = ubicado;
        this.posicion = posicion;
        this.orientacion = orientacion;
    }

    /** This method translate the entity into machine code.
     * @return a string.
     */
    public String mecanizate() {
```

```
63          return null;
64
65
66      }
67
68      /** This method generates a jump between two non connected entities.
69       * @return a string.
70       */
71      public String saltarASiguiente() {
72          String linea = GCode.avanceRapido(this);
73          return linea;
74      }
75
76      /** This method translate an entity into drill code.
77       * @return a string.
78       */
79      public String taladrate() {
80
81          return null;
82      }
83
84      /** This method calculates the compensation of an entity
85       * @return another entity.
86       */
87      public datos compensate() {
88
89          return null;
90      }
91      /** This method calculates the equation that defines an entity.
92       * @param radioHerramienta is the tool diameter
93       * @return an equation.
94       */
95      public EcuacionEntidad calculaTuEcuacion(double radioHerramienta) {
96
97          return null;
98      }
99
100     /** This method compensates a given equation.
101      * @param ecuacion is an equation
102      * @param radioHerramienta is the tool radius.
103      * @param elemento is an entity.
104      * @return an equation.
105      */
106     public EcuacionEntidad compensaTuEcuacion(EcuacionEntidad ecuacion, double
    radioHerramienta, datos elemento) {
107
108
109         return null;
110     }
111
112     /** This method replaces the end point of an entity.
113      * @param interseccion
114      * @return an entity.
115      */
116     public datos cambiarCoordenadaFinal(Coordenadas interseccion) {
117         if(this.orientacion==0){
118             this.FinalX=interseccion.x;
119             this.FinalY=interseccion.y;
120         }else{
121             this.ComienzoX=interseccion.x;
122             this.ComienzoY=interseccion.y;
123         }
```

```
124         return this;
125     }
126     /** This method replaces the start point of an entity.
127      * @param interseccion is a point.
128      * @return an entity.
129      */
130     public datos cambiarCoordenadaInicial(Coordenadas interseccion) {
131         if(this.orientacion==0){
132             this.ComienzoX=interseccion.x;
133             this.ComienzoY=interseccion.y;
134         }else{
135             this.FinalX=interseccion.x;
136             this.FinalY=interseccion.y;
137 
138         }
139         return this;
140 
141     }
142 
143     /** This method calculates the offset of an equation
144      * @param radioHerramienta is the tool radius.
145      * @return an equation.
146      */
147     public EcuacionEntidad calculaTuEcuacionDesplazada(double radioHerramienta) {
148 
149         return null;
150     }
151 
152     /** This method calculates the new start and end point of an entity with compensation.
153      * @param radioHerramienta is the tool radius.
154      * @return an entity.
155      */
156     public datos calculaTusCoordenadasCompensadas(double radioHerramienta) {
157 
158         return null;
159     }
160 
161     /** This method calculates a point over an offset line
162      * @param radioHerramienta is the tool radius.
163      * @param original is an equation..
164      * @return a point.
165      */
166     public Coordenadas ObtenerPuntoSobreRectaCompensada(double radioHerramienta,
167             EcuacionRecta original) {
168 
169         return null;
170     }
171 
172     /** This method calculates the displacement of an entity
173      * @param centro is a point.
174      * @return an entity.
175      */
176     public datos desplazateAlCentro(DatosCirculo centro) {
177 
178     return null;
179     }
180 
181 }
182 
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggDatos;
14
15 import cggGCode.GCode;
16 import cggGCode.compensacionContorno;
17
18 /**
19  * This class defines the structure of DatosArcos
20  * DatosArcos is a data structure that stores the parameters needed to generate an arc.
21  * @author: Celeste G. Guagliano
22  * @version: 13/01/15
23  *
24  */
25
26 public class DatosArcos extends datos {
27     public double radio, Xcentro, Ycentro, AngI, AngF;
28
29     public DatosArcos(double d, double e, double f, double g, double h,
30             double i, double j, double k, double l, int _color,
31             boolean ubicado, int posicion, int o) {
32         super(d, e, f, g, _color, ubicado, posicion, o);
33         this.Xcentro = h;
34         this.Ycentro = i;
35         this.radio = j;
36         this.AngI = k;
37         this.AngF = l;
38
39     }
40
41     /** This method translate the entity into machine code.
42      * @return a string.
43      */
44     public String mecanizate() {
45         String linea = GCode.avanceSegmentoCircular(this);
46         return linea;
47     }
48
49     /** This method calculates the equation that defines an entity.
50      * @param radioHerramienta is the tool radius.
51      * @return an equation.
52      */
53     public EcuacionCircunferencia calculaTuEcuacion(double radioHerramienta){
54         EcuacionCircunferencia
  circunferencia=compensacionContorno.calcularEcuacionCircunferencia(this,radioHerramienta);
55         return circunferencia;
56     }
57
58     /** This method compensates a given equation.
59      * @param ecuacion is an equation.
60      * @param radioHerramienta is the tool radius.
61      * @param elemento is an entity.
62      * @return a equation.
63      */
64     public EcuacionEntidad compensaTuEcuacion(EcuacionEntidad ecuacion,double
  radioHerramienta, datos elemento) {
65         EcuacionCircunferencia compensada=(EcuacionCircunferencia)ecuacion;
66         return ecuacion;
67     }
68
69     /** This method gets the start angle of an arc
70      * @return an angle.
```

```java
     */
    public double obtenerAnguloInicial() {
        double angulo=0;
        if (this.orientacion==0){
            angulo=this.AngI;
        }else
            angulo=this.AngF;
        return angulo;
    }

    /** This method gets the final angle of an arc
     * @return an angle.
     */
    public double obtenerAnguloFinal() {
        double angulo=0;
        if (this.orientacion==0){
            angulo=this.AngF;
        }else
            angulo=this.AngI;
        return angulo;
    }

    /** This method calculates the displacement of an entity
     * @param centro is a point.
     * @return an entity.
     */
    public DatosArcos desplazateAlCentro(DatosCirculo centro) {
        this.ComienzoX=(double)
  FormatoNumeros.formatearNumero(this.ComienzoX-centro.CentroX);
        this.FinalX=(double) FormatoNumeros.formatearNumero(this.FinalX-centro.CentroX);
        this.ComienzoY=(double)
  FormatoNumeros.formatearNumero(this.ComienzoY-centro.CentroY);
        this.FinalY=(double) FormatoNumeros.formatearNumero(this.FinalY-centro.CentroY);
        this.Xcentro=(double) FormatoNumeros.formatearNumero(this.Xcentro-centro.CentroX);
        this.Ycentro=(double) FormatoNumeros.formatearNumero(this.Ycentro-centro.CentroY);
        return this;
    }

}

```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggDatos;
14
15 import java.util.Hashtable;
16
17 import cggColeccion.ColeccionFunciones;
18 import cggGCode.GCode;
19 import cggGCode.compensacionContorno;
20 import cggTablas.Tabla;
21
22 /**
23  * This class defines the structure of DatosCirculo.
24  * DatosCirculo is a data structure that stores the parameters needed to generate a Circle.
25  * @author: Celeste G. Guagliano
26  * @version: 13/01/15
27  *
28  */
29
30 public class DatosCirculo extends datos {
31
32     public double CentroX, CentroY, Radio;
33
34     public DatosCirculo(double d, double e, double f, double g, double h,
35             double i, double j, int _color, boolean ubicado, int posicion, int o) {
36         super(d, e, f, g, _color, ubicado, posicion, o);
37         CentroX = h;
38         CentroY = i;
39         Radio = j;
40     }
41
42     public String mecanizate() {
43         String linea = GCode.avanceCicular(this);
44         return linea;
45     }
46
47     public String taladrate() {
48         String linea = GCode.coordenadasTaladro(this);
49         return linea;
50     }
51
52     public EcuacionCircunferencia calculaTuEcuacion(double radioHerramienta){
53         EcuacionCircunferencia
   circunferencia=compensacionContorno.calcularEcuacionCircunferencia(this,radioHerramienta);
54         return circunferencia;
55
56 }
57     public EcuacionEntidad compensaTuEcuacion(EcuacionEntidad ecuacion,double
   radioHerramienta, datos elemento) {
58         EcuacionCircunferencia compensada=(EcuacionCircunferencia)ecuacion;
59         compensada.Radio=compensada.Radio+radioHerramienta;
60         return compensada;
61     }
62
63     public static DatosCirculo obtenerCentro() {
64         Hashtable ListaCentro = new Hashtable();
65         int colorCentro = 1;
66         DatosCirculo centro=null;
67             ListaCentro = ColeccionFunciones.ObtenerSubconjunto(
68             Tabla.ListaEntidades, colorCentro);
69             if(ListaCentro.size()==1 && (ListaCentro.get(1) instanceof DatosCirculo) ){
70                 centro=(DatosCirculo)ListaCentro.get(1);
```

```
71              }
72              return centro;
73      }
74      public DatosCirculo desplazateAlCentro(DatosCirculo centro) {
75          this.ComienzoX=(double)
  FormatoNumeros.formatearNumero(this.ComienzoX-centro.CentroX);
76          this.FinalX=(double) FormatoNumeros.formatearNumero(this.FinalX-centro.CentroX);
77          this.ComienzoY=(double)
  FormatoNumeros.formatearNumero(this.ComienzoY-centro.CentroY);
78          this.FinalY=(double) FormatoNumeros.formatearNumero(this.FinalY-centro.CentroY);
79          this.CentroX=(double) FormatoNumeros.formatearNumero(this.CentroX-centro.CentroX);
80          this.CentroY=(double) FormatoNumeros.formatearNumero(this.CentroY-centro.CentroY);
81          return this;
82      }
83  }
84
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggDatos;
14
15
16 import cggColeccion.ColeccionFunciones;
17 import cggGCode.GCode;
18 import cggGCode.compensacionContorno;
19
20 /**
21  * This class defines the structure of DatosLinea
22  * DatosLinea is a data structure that stores the parameters needed to generate a line.
23  * @author: Celeste G. Guagliano
24  * @version: 13/01/15
25  *
26  */
27
28 public class DatosLinea extends datos {
29
30     public DatosLinea(double d, double e, double f, double g, int _color,
31             boolean ubicado, int posicion, int o) {
32
33         super(d, e, f, g, _color, ubicado, posicion, o);
34     }
35
36     public String mecanizate() {
37         String linea = GCode.avanceLineal(this);
38         return linea;
39     }
40     public EcuacionEntidad calculaTuEcuacion(double herramienta){
41         EcuacionRecta
  ecuacion=compensacionContorno.calcularEcuacionRecta(this,herramienta);
42         return ecuacion;
43     }
44
45     public Coordenadas ObtenerPuntoSobreRectaCompensada(double radioHerramienta,
46             EcuacionRecta original) {
47         double A=0;
48         double B=0;
49         double C=0;
50         double d=radioHerramienta;
51         Coordenadas iniciales=ColeccionFunciones.ObtenerCoordenadaInicioEntidad(this);
52         Coordenadas finales=ColeccionFunciones.ObtenerCoordenadaFinEntidad(this);
53         Coordenadas punto=new Coordenadas(iniciales.x,iniciales.y);
54         if(iniciales.x!=finales.x){
55             A=original.B*original.B;
56             B=2*original.B*(original.A*punto.x+original.C);
57             C=Math.pow((original.A*punto.x+original.C),2)-d*d*
  (original.A*original.A+original.B*original.B);
58             double y1=(-B+Math.sqrt(B*B-4*A*C))/2*A;
59             double y2=(-B-Math.sqrt(B*B-4*A*C))/2*A;
60             if(finales.x>iniciales.x){
61                 if(y1>y2){
62                     punto.y=y1;
63                 }else{
64                     punto.y=y2;
65                 }
66             }else{
67                 if(y1>y2){
68                     punto.y=y2;
69                 }else{
70                     punto.y=y1;
```

```
                }
            }

        }else{
            A=original.A*original.A;
            B=2*original.A*(original.B*punto.y+original.C);
            C=Math.pow((original.B*punto.y+original.C), 2)-d*d*
  (original.A*original.A+original.B*original.B);
            double x1=(-B+Math.sqrt(B*B-4*A*C))/2*A;
            double x2=(-B-Math.sqrt(B*B-4*A*C))/2*A;
            if(finales.y>iniciales.y){
                if(x1<x2){
                    punto.x=x1;
                }else{
                    punto.x=x2;
                }
            }else{
                if(x1<x2){
                    punto.x=x2;
                }else{
                    punto.x=x1;
                }
            }

        }
        return punto;
        }


    public DatosLinea desplazateAlCentro(DatosCirculo centro) {
    this.ComienzoX=(double) FormatoNumeros.formatearNumero(this.ComienzoX-centro.CentroX);
    this.FinalX=(double) FormatoNumeros.formatearNumero(this.FinalX-centro.CentroX);
    this.ComienzoY=(double) FormatoNumeros.formatearNumero(this.ComienzoY-centro.CentroY);
    this.FinalY=(double) FormatoNumeros.formatearNumero(this.FinalY-centro.CentroY);
    return this;
    }


}

```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggDatos;
14
15 /**
16  * This class defines the structure of EcuacionCircunferencia.
17  * EcuacionCircunferencia is a data structure that stores the parameters of the equation of
   a given circle.
18  * @author: Celeste G. Guagliano
19  * @version: 13/01/15
20  *
21  */
22
23 public class EcuacionCircunferencia extends EcuacionEntidad{
24     public double centroX;
25     public double centroY;
26     public double Radio;
27
28
29     public EcuacionCircunferencia(double cx, double cy,double r) {
30             this.centroX = cx;
31             this.centroY = cy;
32             this.Radio = r;
33
34         }
35
36
37 }
38
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggDatos;
14
15 /**
16  * This class defines the structure of EcuacionRecta.
17  * EcuacionRecta is a data structure that stores the parameters of the equation of a given
   line.
18  * @author: Celeste G. Guagliano
19  * @version: 13/01/15
20  *
21  */
22 public class EcuacionRecta extends EcuacionEntidad {
23     public double A;
24     public double B;
25     public double C;
26
27
28     public EcuacionRecta(double A, double B,double C) {
29         this.A = A;
30         this.B = B;
31         this.C = C;
32
33         }
34     }
35
36
37
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggDatos;
14
15 import java.text.DecimalFormat;
19
20 /**
21  * This class defines the number's format.
22  * @author: Celeste G. Guagliano
23  * @version: 13/01/15
24  *
25  */
26 public class FormatoNumeros {
27
28     /** This method sets the decimal format for a number.
29      * @param numero is a number.
30      * @return a number.
31      */
32     public static Object formatearNumero(double numero){
33         DecimalFormatSymbols simbolos = new DecimalFormatSymbols();
34         simbolos.setDecimalSeparator('.');
35         DecimalFormat formateador = new DecimalFormat("#####.###",simbolos);
36         try{
37         numero=Double.parseDouble(formateador.format(numero));
38         return numero;
39         }catch(NumberFormatException ex){
40              JOptionPane.showMessageDialog(null, "Imposible generar el mecanizado
   solicitado","Error" ,JOptionPane.ERROR_MESSAGE);
41             };
42         return null;
43 }
44 }
45
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13
14 package cggDatos;
15 /**
16  * This class defines the structure of Herramienta.
17  * Herramienta is a data structure that stores the parameters of a given tool.
18  * @author: Celeste G. Guagliano
19  * @version: 13/01/15
20  *
21  */
22 public class Herramienta {
23     public double Diametro = 0;
24     public double Avance = 0;
25     public double Velocidad = 0;
26     public double Pasada = 0;
27     public double Numero = 0;
28     public double Zseguro=0;
29     public double Zcambio=0;
30
31     public Herramienta(double num, double dia, double ava, double velo,
32             double pasa,double zseg,double zcam) {
33         this.Numero = num;
34         this.Diametro = dia;
35         this.Velocidad = velo;
36         this.Avance = ava;
37         this.Pasada = pasa;
38         this.Zseguro=zseg;
39         this.Zcambio=zcam;
40     }
41
42 }
43
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13
14 package cggGCode;
15
16 import java.util.Enumeration;
33
34 /**
35  * This class implements the necessary algorithms to calculate a new contour compensating
   the radius of the selected tool to operate.
36  * @author: Celeste G. Guagliano
37  * @version: 13/01/15
38  *
39  */
40 public class compensacionContorno {
41
42     /** Method to calculate the equation of the line
43      * @param datosLinea it's the line generated by the equation wanted
44      * @param herramienta it's the tool involve in the process
45      * @return a line equation.
46      */
47     public static EcuacionRecta calcularEcuacionRecta(datos datosLinea, double herramienta)
   {
48
49         double A=0;
50         double B=0;
51         double C=0;
52         double pendiente=0;
53         double terminoIndependiente=0;
54         Coordenadas
   iniciales=ColeccionFunciones.ObtenerCoordenadaInicioEntidad(datosLinea);
55         Coordenadas finales=ColeccionFunciones.ObtenerCoordenadaFinEntidad(datosLinea);
56         Coordenadas punto=new Coordenadas(0,0);
57
58         double deltaX=(finales.x-iniciales.x);
59         double deltaY=(finales.y-iniciales.y);
60         if(deltaX!=0){
61         B=1;
62         pendiente=deltaY/deltaX;
63         A=-pendiente;
64         C=pendiente*iniciales.x-iniciales.y;
65         }else{
66         B=0;
67         A=-1;
68         C=iniciales.x;
69         }
70         EcuacionRecta original=new EcuacionRecta(A,B,C);
71         punto=datosLinea.ObtenerPuntoSobreRectaCompensada(herramienta,original);
72         EcuacionRecta compensada=new EcuacionRecta(0,0,0);
73         compensada=calcularRectaParalela(original,punto);
74         return compensada;
75     }
76
77     /** Method to calculate the equation of parallel line to another given line.
78      * @param original is the given line.
79      * @param punto is a point include in the parallel line.
80      * @return a line equation.
81      */
82         private static EcuacionRecta calcularRectaParalela(EcuacionRecta original,
83         Coordenadas punto) {
84         EcuacionRecta compensada= new EcuacionRecta(original.A,original.B,0);
85         compensada.C=-compensada.A*punto.x-original.B*punto.y;
```

```
 86         return compensada;
 87 }
 88 
 89 
 90         /** Method to calculate the equation a circumference
 91          * @param datosCirculo is an entity.
 92          * @param radioHerramienta is the tool radius.
 93          * @return a circumference equation.
 94          */
 95         public static EcuacionCircunferencia calcularEcuacionCircunferencia(datos
    datosCirculo, double radioHerramienta) {
 96             EcuacionCircunferencia ecuacion=new EcuacionCircunferencia(0,0,0);
 97             if(datosCirculo instanceof DatosCirculo){
 98                 DatosCirculo dato=(DatosCirculo)datosCirculo;
 99                 ecuacion.centroX=dato.CentroX;
100                 ecuacion.centroY=dato.CentroY;
101                 ecuacion.Radio=dato.Radio+radioHerramienta;
102             }else{
103                 DatosArcos datos=(DatosArcos)datosCirculo;
104                 ecuacion.centroX=datos.Xcentro;
105                 ecuacion.centroY=datos.Ycentro;
106                 if(datos.orientacion==0){
107                     ecuacion.Radio=datos.radio-radioHerramienta;
108                 }else{
109                     ecuacion.Radio=datos.radio+radioHerramienta;
110                 }
111             }
112         return ecuacion;
113     }
114 
115         /** Method to calculate an intersection point between two equations.
116          * @param listaOptimizada a list of entities.
117          * @param listaEcuaciones a list of equations.
118          * @return a list.
119          */
120     public static Hashtable intersectarEcuacionesCompensadas(
121             Hashtable listaOptimizada, Hashtable listaEcuaciones) {
122         Hashtable listaCompensada=new Hashtable();
123         int j=0;
124         for(int i=1;i<=listaEcuaciones.size();i++){
125             EcuacionEntidad ecuacion1=(EcuacionEntidad)listaEcuaciones.get(i);
126             EcuacionEntidad ecuacion2=null;
127             datos elemento1=(datos)listaOptimizada.get(i);
128             datos elemento2=null;
129             if(i<listaEcuaciones.size()){
130                 j=i+1;
131                 ecuacion2=(EcuacionEntidad)listaEcuaciones.get(j);
132                 elemento2=(datos)listaOptimizada.get(j);
133             }else{
134                 j=1;
135                 ecuacion2=(EcuacionEntidad)listaEcuaciones.get(j);
136                 elemento2=(datos)listaOptimizada.get(j);
137             }
138             if(elemento1!=elemento2){
139             listaCompensada=OptimizacionMetodo2.intersectarEntidades(i,j,ecuacion1,elemento1
    ,ecuacion2,elemento2,listaCompensada);
140             }else{
141                 listaCompensada=compensarCircunferencia(i,(EcuacionCircunferencia)ecuacion1,
    (DatosCirculo)elemento1);
142             }
143         }
144 
```

```
145         return listaCompensada;
146     }
147 
148 
149     /** Method to gets a compensated circumference.
150      * @param key is the position in the hashtable.
151      * @param ecuacion1 is the circumference equation.
152      * @param elemento1 is the circumference to compensate.
153      * @return a list.
154      */
155     private static Hashtable compensarCircunferencia(int key,
156             EcuacionCircunferencia ecuacion1, DatosCirculo elemento1) {
157         Hashtable listaCompensada=new Hashtable();
158         elemento1.ComienzoX=(double)
    FormatoNumeros.formatearNumero(elemento1.CentroX-ecuacion1.Radio);
159         elemento1.FinalX=(double)
    FormatoNumeros.formatearNumero(elemento1.CentroX+ecuacion1.Radio);
160         elemento1.Radio=(double) FormatoNumeros.formatearNumero(ecuacion1.Radio);
161         listaCompensada.put(key, elemento1);
162         return listaCompensada;
163     }
164 
165 
166     /** Method to calculate the intersection point between two lines.
167      * @param ecuacion1 is the first line equation.
168      * @param ecuacion2 is the second line equation.
169      * @return a point.
170      */
171     public static Coordenadas intersectarRectas(EcuacionRecta ecuacion1,
172             EcuacionRecta ecuacion2) {
173         Coordenadas interseccion=new Coordenadas(0,0);
174         if(ecuacion1.B!=0){
175             if(ecuacion2.B!=0){
176                 interseccion.x=(ecuacion1.C/ecuacion1.B-ecuacion2.C/ecuacion2.B)/
    (ecuacion2.A/ecuacion2.B-ecuacion1.A/ecuacion1.B);
177                 interseccion.y=-ecuacion1.A/ecuacion1.B*interseccion.x-ecuacion1.C/ecuacion1
    .B;
178             }else{
179                 interseccion.x=-ecuacion2.C/ecuacion2.A;
180                 interseccion.y=-ecuacion1.A/ecuacion1.B*interseccion.x-ecuacion1.C/ecuacion1
    .B;
181             }
182         }else{
183             interseccion.x=-ecuacion1.C/ecuacion1.A;
184             interseccion.y=-ecuacion2.A/ecuacion2.B*interseccion.x-ecuacion2.C/ecuacion2.B;
185         }
186         interseccion.x=(double) FormatoNumeros.formatearNumero(interseccion.x);
187         interseccion.y=(double) FormatoNumeros.formatearNumero(interseccion.y);
188         return interseccion;
189     }
190 
191 
192     /** Method to calculate the point of intersection between a line and a circumference.
193      * @param ecuacion1 is the line's equation.
194      * @param ecuacion2 is the circumference's equation.
195      * @param elemento2 is the Arc's data.
196      * @return a point.
197      */
198     public static Coordenadas intersectarArcoYRecta(EcuacionRecta ecuacion1,
199             EcuacionCircunferencia ecuacion2, DatosArcos elemento2) {
200         Coordenadas interseccion=new Coordenadas(0,0);
201         Coordenadas finArco=ColeccionFunciones.ObtenerCoordenadaFinEntidad(elemento2);
```

```
202         if(ecuacion1.A==0){
203             interseccion=intersectarArcoConRectaHorizontal(ecuacion1,ecuacion2,finArco);
204         }else if(ecuacion1.B==0){
205             interseccion=intersectarArcoConRectaVertical(ecuacion1,ecuacion2,finArco);
206         }else{
207             interseccion=intersectarRectaConArcoCasoGeneral(ecuacion1,ecuacion2,finArco);
208         }
209         return interseccion;
210     }
211 
212     /** Method to calculate the point of intersection between a line and a circumference.
213      * @param ecuacion1 is the line's equation.
214      * @param ecuacion2 is the circumference's equation.
215      * @param elemento2 is the Arc's data.
216      * @return a point.
217      */
218     public static Coordenadas intersectarRectaYArco(EcuacionRecta ecuacion1,
219         EcuacionCircunferencia ecuacion2, DatosArcos elemento2) {
220         Coordenadas interseccion=new Coordenadas(0,0);
221         Coordenadas
    inicioArco=ColeccionFunciones.ObtenerCoordenadaInicioEntidad(elemento2);
222         if(ecuacion1.A==0){
223             interseccion=intersectarArcoConRectaHorizontal(ecuacion1,ecuacion2,inicioArco);
224         }else if(ecuacion1.B==0){
225             interseccion=intersectarArcoConRectaVertical(ecuacion1,ecuacion2,inicioArco);
226         }else{
227             interseccion=intersectarRectaConArcoCasoGeneral(ecuacion1,ecuacion2,inicioArco);
228         }
229         return interseccion;
230         }
231 
232     /** Method to calculate the point of intersection between a line and a circumference.
233      * @param ecuacion1 is the line's equation.
234      * @param ecuacion2 is the circumference's equation.
235      * @param elemento2 is the Arc's data.
236      * @return a point.
237      */
238         private static Coordenadas intersectarRectaConArcoCasoGeneral(
239             EcuacionRecta ecuacion1, EcuacionCircunferencia ecuacion2,
240             Coordenadas interseccionArcoOriginal) {
241             JOptionPane.showMessageDialog(null, "Lo siento, la versión actual no permite
    generar el contorneado seleccionado", "Error", JOptionPane.ERROR_MESSAGE);
242             return null;
243     }
244 
245         /** Method to calculate the point of intersection between a  vertical line and a
    circumference.
246          * @param ecuacion1 is the line's equation.
247          * @param ecuacion2 is the circumference's equation.
248          * @param elemento2 is the Arc's data.
249          * @return a point.
250          */
251         private static Coordenadas intersectarArcoConRectaVertical(
252             EcuacionRecta ecuacion1, EcuacionCircunferencia ecuacion2,
253             Coordenadas interseccionArcoOriginal) {
254         Coordenadas interseccion=new Coordenadas(0,0);
255         interseccion.x=-ecuacion1.C/ecuacion1.A;
256         System.out.println(ecuacion2.Radio+"*"+ecuacion2.Radio+"-Math.pow"+interseccion.x+"
    -"+ecuacion2.centroX);
257         double parametroAlCuadrado=
    (ecuacion2.Radio*ecuacion2.Radio-Math.pow(interseccion.x-ecuacion2.centroX,2));
258         double parametro=0;
```

```
259         if(Math.abs(parametroAlCuadrado)>0.0001){
260             parametro=Math.sqrt(parametroAlCuadrado);
261         }
262         double y1=ecuacion2.centroY+parametro;
263         double y2=ecuacion2.centroY-parametro;
264         interseccion.y=seleccionarPuntoY(ecuacion1,ecuacion2,y1,y2,interseccionArcoOriginal)
    ;
265         interseccion.x=(double) FormatoNumeros.formatearNumero(interseccion.x);
266         interseccion.y=(double) FormatoNumeros.formatearNumero(interseccion.y);
267         return interseccion;
268     }
269 
270         /** Method to select an intersection point of two possibilities.
271          * @param ecuacion2 is the circumference's equation
272          * @param y1 is the first possibly intersection point calculated.
273          * @param y2 is the second possibly intersection point calculated.
274          * @param interseccionArcoOriginal is the original point of intersection between an
    arc and another entity.
275          * @return a point.
276          */
277         private static double seleccionarPuntoY(EcuacionRecta ecuacion1,
278                 EcuacionCircunferencia ecuacion2, double y1, double y2,
279                 Coordenadas interseccionArcoOriginal) {
280             double y=0;
281             double cumple1=ecuacion1.A*
    (-ecuacion1.B*y1/ecuacion1.A-ecuacion1.C/ecuacion1.A)+ecuacion1.B*y1+ecuacion1.C;
282             double cumple2=ecuacion1.A*
    (-ecuacion1.B*y2/ecuacion1.A-ecuacion1.C/ecuacion1.A)+ecuacion1.B*y2+ecuacion1.C;
283             if(cumple1==0){
284                 y=y1;
285             }else{
286                 if(cumple2==0){
287                     y=y2;
288                 }
289             }
290             return y;
291         }
292 
293         /** Method to calculate the point of intersection between an horizontal line and a
    circumference.
294          * @param ecuacion1 is the line's equation.
295          * @param ecuacion2 is the circumference's equation.
296          * @param elemento2 is the Arc's data.
297          * @return a point.
298          */
299         private static Coordenadas intersectarArcoConRectaHorizontal(
300             EcuacionRecta ecuacion1, EcuacionCircunferencia ecuacion2,
301             Coordenadas interseccionArcoOriginal) {
302         Coordenadas interseccion=new Coordenadas(0,0);
303         interseccion.y=-ecuacion1.C/ecuacion1.B;
304         double parametro=0;
305         if(FormatoNumeros.formatearNumero(ecuacion2.Radio*ecuacion2.Radio)!
    =FormatoNumeros.formatearNumero(Math.pow(interseccion.y-ecuacion2.centroY,2))){
306         parametro=Math.sqrt(ecuacion2.Radio*ecuacion2.Radio-Math.pow(interseccion.y-ecuacio
    n2.centroY,2));
307         }
308         double x1=ecuacion2.centroX+parametro;
309         double x2=ecuacion2.centroX-parametro;
310         interseccion.x=seleccionarPuntoX(ecuacion2,x1,x2,interseccionArcoOriginal);
311         interseccion.x=(double) FormatoNumeros.formatearNumero(interseccion.x);
312         interseccion.y=(double) FormatoNumeros.formatearNumero(interseccion.y);
313         return interseccion;
```

```
314     }
315
316
317         /** Method to select an intersection point of two possibilities.
318          * @param ecuacion2 is the circumference's equation
319          * @param x1 is the first possibly intersection point calculated.
320          * @param x2 is the second possibly intersection point calculated.
321          * @param interseccionArcoOriginal is the original point of intersection between an
    arc and another entity.
322          * @return a point.
323          */
324         private static double seleccionarPuntoX(
325                 EcuacionCircunferencia ecuacion2, double x1, double x2,
326                 Coordenadas interseccionArcoOriginal) {
327             double x=0;
328             if(interseccionArcoOriginal.x<ecuacion2.centroX){
329                 if(x1<ecuacion2.centroX){
330                     x=x1;
331                 }else{
332                     x=x2;
333                 }
334             }else{
335                 if(x1>ecuacion2.centroX){
336                     x=x1;
337                 }else{
338                     x=x2;
339                 }
340             }
341             return x;
342         }
343
344         /** Method to calculate the point of intersection between two circumferences.
345          * @param ecuacion1 is the first circumference's equation.
346          * @param ecuacion2 is the second circumference's equation.
347          * @param elemento1 is the first Arc's data.
348          * @param elemento1 is the second Arc's data.
349          * @return a point.
350          */
351     public static Coordenadas intersectarArcos(EcuacionCircunferencia ecuacion1,
352             EcuacionCircunferencia ecuacion2, DatosArcos elemento1, DatosArcos elemento2) {
353             JOptionPane.showMessageDialog(null, "Lo siento, la versión actual no permite
    generar el contorneado seleccionado", "Error", JOptionPane.ERROR_MESSAGE);
354             return null;
355     }
356
357
358     /** Method to calculate a list of equations from a list of entities.
359      * @param ecuacion2 is the circumference's equation
360      * @param y1 is the first possibly intersection point calculated.
361      * @param y2 is the second possibly intersection point calculated.
362      * @param interseccionArcoOriginal is the original point of intersection between an arc
    and another entity.
363      * @return a point.
364      */
365     public static Hashtable ObtenerListaEcuaciones(
366         Hashtable listaEntidadesCompensadas, double radioHerramienta) {
367         Hashtable listaEcuaciones=new Hashtable();
368         EcuacionEntidad ecuacion= new EcuacionEntidad();
369         for(int i=1; i<listaEntidadesCompensadas.size()+1;i++){
370             datos elemento=(datos)listaEntidadesCompensadas.get(i);
371             ecuacion=elemento.calculaTuEcuacion(radioHerramienta);
372             listaEcuaciones.put(i, ecuacion);
```

```
373         }
374         return listaEcuaciones;
375     }
376 
377 }
378 
379 
380 
381 
382 
383 
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggGCode;
14
15 import java.awt.Color;
44 /**
45  * This class implements the necessary algorithms to translate all the collected data into
   G-Code
46  * @author: Celeste G. Guagliano
47  * @version: 13/01/15
48  *
49  */
50
51 public class GCode {
52     public static String nombreA;
53     public static String nombreArchivo;
54     public static String encabezado;
55     public static String cambioHerramienta;
56     public static String definirAvance;
57     public static String avanceRapido;
58     public static String avanceRapidoZ;
59     public static String avanceLineal;
60     public static String avanceSegmentoCircularHorario;
61     public static String avanceSegmentoCircularAntihorario;
62     public static String avanceCircularHorario;
63     public static String avanceCircularAntihorario;
64     public static String taladrado;
65     public static String taladradoProfundo;
66     public static String coordenadas;
67     public static String cancelarCicloFijo;
68     public static String terminacion;
69     public static String llamadaSubprograma;
70     public static String retornoSubprograma;
71     public static String rampaZ;
72     public static boolean subprograma;
73     public static String avanceLinealZ;
74     public static String nombreSubprograma;
75     public static String coordenadaAbsoluta;
76     public static String coordenadaRelativa;
77     public static String Ruta;
78     public static String properties;
79     public static int nroSubprograma=0;
80     public static String llamadaSubprog;
81     public static JTextArea grabado = new JTextArea();
82     public static JTextArea contorneado = new JTextArea();
83     public static JTextArea taladro = new JTextArea();
84     public static JTextArea plano=new JTextArea();
85
86
87     /** Method to generate the drilling process.
88      * @param mecanizarRasgo is an instance of the method used to machining the selected
   feature.
89      * @param Lista is the entities's list to be machined
90      * @param Herramientas is the tool's list
91      * @param profu is the deepness of the feature.
92      * @param principal is the text area where the code of the main file is generated.
93      * @return the drilling process code.
94      */
95     public static JTextArea GenerarTaladrado(GCode mecanizarRasgo, Hashtable Lista,
96         Hashtable Herramientas, double profu, JTextArea principal) {
97         Hashtable TaladradoOptimizado = ObtenerTaladradoOptimizado(Lista);
98         int P = 1;
```

```
 99          datos elemento = (datos) TaladradoOptimizado.get(1);
100          double profuRasgo = Double.parseDouble(DXF_Loader.profTaladro.getText());
101          Herramienta herramienta = (Herramienta) Herramientas.get("Taladrado");
102          double zcambio=herramienta.Zcambio;
103          double zsafe=herramienta.Zseguro;
104          GCodeMetodoTaladrado.inicializarTaladrado(elemento,Herramientas,principal,profuRasg
     o);
105          for (int i = 2; i <= TaladradoOptimizado.size(); i++) {
106              elemento = (datos) TaladradoOptimizado.get(i);
107              String linea = elemento.taladrate();
108              taladro.append(linea);
109          }
110          finalizarTaladrado(taladro);
111          principal = finalizar(taladro, principal, P,zcambio);
112          return principal;
113
114      }
115
116      /** Method to end the drilling process
117       * @param taladrado is the text area where the drill process code is generated.
118       * @return the drilling process code.
119       */
120      private static void finalizarTaladrado(JTextArea taladrado) {
121          taladrado.append(cancelarCicloFijo);
122      }
123
124
125
126
127      /** Method to generate the contouring process.
128       * @param mecanizarRasgo is an instance of the method used to machining the selected
     feature.
129       * @param Lista is the entities's list to be machined
130       * @param Herramientas is the tool's list
131       * @param profu is the deepness of the feature.
132       * @param principal is the text area where the code of the main file is generated.
133       * @return the contouring process code.
134       */
135      public JTextArea GenerarContorneado(GCode mecanizarRasgo, Hashtable Lista,
136          Hashtable Herramientas, double profu, JTextArea principal) {
137          Herramienta herramienta = (Herramienta) Herramientas.get("Contorneado");
138          Hashtable ContornoOptimizado=new Hashtable();
139          ContornoOptimizado = ObtenerContorneadoOptimizado(Lista,herramienta);
140          double Zcambio=herramienta.Zcambio;
141          double pasada = herramienta.Pasada;
142          double profuRasgo = Double.parseDouble(DXF_Loader.profContorno
143                  .getText());
144          int P = 0;
145          if (pasada > profuRasgo) {
146              pasada = profuRasgo;
147          } else {
148              pasada =(double) FormatoNumeros.formatearNumero(ObtenerPasada(pasada,
     profuRasgo));
149          }
150              P = ObtenerRepeticiones(pasada, profuRasgo);
151              inicializarRasgo(principal, herramienta,
152              (datos) ContornoOptimizado.get(1), profuRasgo,0);
153              contorneado = mecanizarRasgo.mecanizarLista(ContornoOptimizado, contorneado,
     pasada, P,
154                      herramienta);
155          principal = finalizar(contorneado, principal, 1,Zcambio);
156          return principal;
```

```
157
158     }
159
160     private static int ObtenerRepeticiones(double pasada, double profuRasgo) {
161         int P = (int) (profuRasgo / pasada);
162         return P;
163     }
164
165     private static double ObtenerPasada(double pasada, double profuRasgo) {
166         double cantidad = 0;
167         if (pasada <= profuRasgo) {
168             cantidad = profuRasgo / pasada;
169             if (cantidad - (int) cantidad > 0) {
170                 pasada = profuRasgo / ((int) cantidad);
171             }
172         } else {
173             pasada = profuRasgo;
174         }
175         pasada=(double) FormatoNumeros.formatearNumero(pasada);
176         return pasada;
177     }
178
179     /** Method to generate the engraving process.
180      * @param mecanizarRasgo is an instance of the method used to machining the selected
    feature.
181      * @param Lista is the entities's list to be machined
182      * @param Herramientas is the tool's list
183      * @param profu is the deepness of the feature.
184      * @param principal is the text area where the code of the main file is generated.
185      * @return the engraving process code.
186      */
187     public JTextArea GenerarGrabado(GCode mecanizarRasgo,Hashtable Lista,
188         Hashtable Herramientas, double profu, JTextArea principal) {
189         Hashtable GrabadoOptimizado = ObtenerGrabadoOptimizado(Lista);
190         Herramienta herramienta = (Herramienta) Herramientas.get("Grabado");
191         double pasada = herramienta.Pasada;
192         double zcambio=herramienta.Zcambio;
193         int P = 0;
194         double profuRasgo = Double.parseDouble(DXF_Loader.profGrabo.getText());
195         if (pasada > profuRasgo) {
196             pasada = profuRasgo;
197         } else {
198             pasada = ObtenerPasada(pasada, profuRasgo);
199         }
200             P = ObtenerRepeticiones(pasada, profuRasgo);
201         inicializarRasgo(principal, herramienta,
202                 (datos) GrabadoOptimizado.get(1), profuRasgo,pasada);
203         grabado = mecanizarRasgo.mecanizarLista(GrabadoOptimizado, grabado, pasada, P,
204                 herramienta);
205         P=1;
206         principal = finalizar(grabado, principal, P,zcambio);
207         return principal;
208
209     }
210
211 /**Method to generate the translate of the entities into Machine code
212  * @param lista is the list of entities
213  * @param rasgo is the text area where the code is written
214  * @param pasada is the depth of the feature
215  * @param repeticiones is the number of times that the subprogram must be repeated if the
    code is generated into a subprogram.
216  * @param herramienta is the tool involved in the process.
```

```
217 * @return the code in a text area.
218 */
219    public JTextArea mecanizarLista(Hashtable lista, JTextArea rasgo,
220            double pasada, int repeticiones, Herramienta herramienta) {
221        return null;
222    }
223
224
225    /** Method to generate the facing process.
226     * @param mecanizarRasgo is an instance of the method used to machining the selected
    feature.
227     * @param Lista is the entities's list to be machined
228     * @param Herramientas is the tool's list
229     * @param profu is the deepness of the feature.
230     * @param principal is the text area where the code of the main file is generated.
231     * @return the facing process code.
232     */
233    public static JTextArea GenerarPlaneado(GCode mecanizarRasgo,Hashtable Lista,
234            Hashtable herramientas, double profu, JTextArea principal) {
235        Hashtable PlaneadoOptimizado = ObtenerPlaneadoOptimizado(Lista);
236        Herramienta herramienta = (Herramienta) herramientas.get("Planeado");
237        double zcambio=herramienta.Zcambio;
238        double pasada = herramienta.Pasada;
239        int P = 0;
240        double profuRasgo = Double.parseDouble(DXF_Loader.profPlano.getText());
241        if (pasada > profuRasgo) {
242            pasada = profuRasgo;
243            } else {
244            pasada = ObtenerPasada(pasada, profuRasgo);
245            }
246            P = ObtenerRepeticiones(pasada, profuRasgo);
247        inicializarRasgo(principal, herramienta,
248                (datos) PlaneadoOptimizado.get(1), profuRasgo,pasada);
249
250        plano = mecanizarRasgo.mecanizarLista(PlaneadoOptimizado, plano, pasada, P,
251                herramienta);
252        P=1;
253        principal = finalizar(plano, principal, P,zcambio);
254        return principal;
255    }
256
257    /**
258     * Method to finish the translation of a particular feature.
259     * @param rasgo is the feature to finish.
260     * @param principal is the main translation file.
261     * @param repeticiones is the number of times that a subprogram must be repeated if the
    feature is generated in a subprogram.
262     * @param zcambio is the security high for the tool's change.
263     * @return the generated code in a texr area.
264     */
265    private static JTextArea finalizar(JTextArea rasgo, JTextArea principal,
266            int repeticiones, double zcambio) {
267        rasgo=chequearContenido(rasgo);
268        if (subprograma == false) {
269            principal.append(rasgo.getText());
270        } else {
271            ++nroSubprograma;
272            llamadaSubprog = llamadaSubprograma.replace("programa",
    Integer.toString(nroSubprograma));
273            llamadaSubprog=llamadaSubprog.replace("repeticiones",Integer.toString(repeticio
    nes));
274            String
```

```
nombreSubprog=nombreSubprograma.replace("nombre",Integer.toString(nroSubprograma));
            String ruta = new String(Ruta + nombreSubprog);
            rasgo.append(retornoSubprograma);
            try {
                BufferedWriter out = new BufferedWriter(new FileWriter(ruta));
                out.write(rasgo.getText());
                out.close();
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null, ex.getMessage());
            }
            principal.append(llamadaSubprog);
        }
        String
linea=coordenadaAbsoluta+avanceRapidoZ.replace("z",Double.toString(zcambio));
        principal.append(linea);
        return principal;
    }

    /**
     * Method to check if there's any code generated in a text area
     * @param rasgo is the text area to check
     * @return the text area checked.
     */
    private static JTextArea chequearContenido(JTextArea rasgo) {
        if(rasgo.getText()==null){
            rasgo.setText("Lo siento, algunos de los elementos que desea mecanizar no son
admitidos aï¿½n por DXF2GCode,"
                    + "Por favor, revise las entidades seleccionadas o modifique el archivo
en su editor CAD para proseguir.");
        }
        return rasgo;
    }

    /**
     * Method to get the optimized list of entities for the face milling.
     * @param listaTocho the face milling list.
     * @return an optimized list of entities.
     */
    public static Hashtable ObtenerPlaneadoOptimizado(Hashtable listaTocho) {
        Hashtable ListaTochoOptimizada = new Hashtable();
        ListaTochoOptimizada = Optimizacion.Optimizacion(listaTocho);
        return ListaTochoOptimizada;
    }

    /**
     * Method to get the optimized list of entities for the contouring.
     * @param listaContorno the contouring list.
     * @param contorneado the tool involved in the process.
     * @return an optimized list of entities.
     */
    public static Hashtable ObtenerContorneadoOptimizado(Hashtable
listaContorno,Herramienta contorneado) {
        Hashtable ListaContornoOptimizada = new Hashtable();
        ListaContornoOptimizada =
OptimizacionMetodo2.Optimizacion(TablaContorno.ListaContorno,contorneado.Diametro/2);
        return ListaContornoOptimizada;
    }

    /** Method to get the optimized list of entities for the engraving.
     * @param listaGrabado is the engraving list.
     * @return an optimized list of entities.
     */
```

```
331     public static Hashtable ObtenerGrabadoOptimizado(Hashtable listaGrabado) {
332         Hashtable ListaGrabadoOptimizada = new Hashtable();
333         ListaGrabadoOptimizada = OptimizacionMetodo1
334                 .Optimizacion(TablaGrabado.ListaGrabado);
335         return ListaGrabadoOptimizada;
336     }
337 
338     /**
339      * Method to get the optimized list of entities for the drilling.
340      * @param listaAgujeros is the drilling list.
341      * @return an optimized list of entities.
342      */
343     public static Hashtable ObtenerTaladradoOptimizado(Hashtable listaAgujeros) {
344         Hashtable ListaAgujerosOptimizada = new Hashtable();
345         ListaAgujerosOptimizada = Optimizacion
346                 .Optimizacion(TablaAgujeros.ListaAgujeros);
347         return ListaAgujerosOptimizada;
348     }
349 
350     /**
351      * Method to generate the initialization of the main translation file.
352      * @param principal is the text area where the code is generated.
353      * @return a text area with the generated code.
354      */
355     public static JTextArea EncabezarPrograma(JTextArea principal) {
356         String nroProg = "0001";
357         String linea=encabezado.replace("numero", nroProg);
358             principal.append(linea);
359         return principal;
360     }
361 
362     /**
363      * Method to save the main file
364      * @param mecanizado is the text area with the main code generated.
365      * @param consola is the console to display the generated code in the user's interface.
366      */
367     public static void archivarMecanizado(JTextArea mecanizado, JTextPane consola) {
368         mecanizado.append(terminacion);
369         int numero=0;
370         File f;
371         String ruta;
372         do{
373         ++numero;
374         nombreA=nombreArchivo.replace("nombre",Integer.toString(numero));
375         ruta = new String(Ruta +nombreA);
376         f = new File(ruta);
377         }while(f.exists());
378         try {
379             BufferedWriter out = new BufferedWriter(new FileWriter(ruta));
380             out.write(mecanizado.getText());
381             out.close();
382         } catch (Exception ex) {
383             JOptionPane.showMessageDialog(null, ex.getMessage());
384         }
385     }
386 
387     /**
388      * Method to initialize a feature
389      * @param principal is the text area of the main program.
390      * @param tool is the tool involved in the process.
391      * @param inicial is the data of the first entity to translate.
392      * @param profRasgo is the depth of the feature.
```

```
393      * @param pasada is the partial depth of the machining process.
394      * @return a text area with the initialized code.
395      */
396     public static JTextArea inicializarRasgo(JTextArea principal,
397             Herramienta tool, datos inicial, double profRasgo, double pasada) {
398         int nroHerramienta = (int) tool.Numero;
399         int velocidadHerramienta = (int) tool.Velocidad;
400         double referencia=tool.Zseguro;
401         double ZcambioHerramienta=tool.Zcambio;
402         double avanceT= tool.Avance;
403         Coordenadas inicio=new Coordenadas(0,0);
404         inicio =ColeccionFunciones.ObtenerCoordenadaInicioEntidad(inicial);
405         String
    linea=cambioHerramienta.replaceAll("herramienta",Integer.toString(nroHerramienta));
406         linea=linea.replace("velocidad", Integer.toString(velocidadHerramienta));
407         linea=linea.replace("x",Double.toString(inicio.x));
408         linea=linea.replace("y",Double.toString(inicio.y));
409         linea=linea.replace("planoRetraccion", Double.toString(referencia));
410         principal.append(linea);
411         linea=definirAvance.replace("referencia",Double.toString(referencia));
412         linea=linea.replace("avance",Double.toString(avanceT));
413         principal.append(linea);
414         linea=avanceLinealZ.replace("z","-"+Double.toString(pasada));
415         principal.append(linea);
416         return principal;
417     }
418 
419 /**
420  * Method to get the data of the machine post-processor
421  * @param postprocesador is the machine info
422  * @param ruta2 is the directory where the file will be saved
423  */
424     public static void prepararPostprocesador(String postprocesador,
425             String ruta2) {
426         Ruta = ruta2;
427         properties = postprocesador;
428         ResourceBundle maquina = ResourceBundle.getBundle(properties);
429         nombreArchivo=maquina.getString("nombreArchivo");
430         encabezado=maquina.getString("encabezado");
431         cambioHerramienta=maquina.getString("cambioHerramienta");
432         definirAvance=maquina.getString("definirAvance");
433         avanceRapido=maquina.getString("avanceRapidoXY");
434         avanceRapidoZ=maquina.getString("avanceRapidoZ");
435         avanceLineal=maquina.getString("avanceLineal");
436         avanceSegmentoCircularHorario=maquina.getString("avanceSegmentoCircularHorario");
437         avanceSegmentoCircularAntihorario=maquina.getString("avanceSegmentoCircularAntihora
    rio");
438         avanceCircularHorario=maquina.getString("avanceCircularHorario");
439         avanceCircularAntihorario=maquina.getString("avanceCircularAntihorario");
440         taladrado=maquina.getString("taladrado");
441         taladradoProfundo=maquina.getString("taladradoProfundo");
442         coordenadas=maquina.getString("coordenadas");
443         cancelarCicloFijo=maquina.getString("finalizarCicloFijo");
444         terminacion=maquina.getString("terminacion");
445         llamadaSubprograma=maquina.getString("llamadaASubprog");
446         retornoSubprograma=maquina.getString("retornoDeSubprograma");
447         subprograma=Boolean.parseBoolean(maquina.getString("subprogramas"));
448         avanceLinealZ=maquina.getString("avanceLinealZ");
449         nombreSubprograma=maquina.getString("nombreSubprograma");
450         coordenadaAbsoluta=maquina.getString("coordenadaAbsoluta");
451         coordenadaRelativa=maquina.getString("coordenadaRelativa");
452         rampaZ=maquina.getString("rampaZ");
```

```
453     }
454
455 /**
456  * Method to generate the linear feed.
457  * @param elemento the line to be translated.
458  * @return a string
459  */
460     public static String avanceLineal(DatosLinea elemento) {
461         String linea = null;
462         Coordenadas dato=ColeccionFunciones.ObtenerCoordenadaFinEntidad(elemento);
463         linea = avanceLineal.replace("x",Double.toString(dato.x));
464         linea=linea.replace("y",Double.toString(dato.y));
465         return linea;
466     }
467
468     /**
469      * Method to generate the fast forward.
470      * @param elemento is the final point of the movement.
471      * @return a string.
472      */
473     public static String avanceRapido(datos elemento) {
474         String linea = null;
475         Coordenadas dato=ColeccionFunciones.ObtenerCoordenadaInicioEntidad(elemento);
476         linea = avanceRapido.replace("x", Double.toString(dato.x));
477         linea=linea.replace("y",Double.toString(dato.y));
478         return linea;
479     }
480
481     /**
482      * Method to generate the circularSegment feed.
483      * @param elemento is the arc to translate.
484      * @return a string.
485      */
486
487     public static String avanceSegmentoCircular(DatosArcos elemento) {
488         String linea = null;
489         Coordenadas dato=ColeccionFunciones.ObtenerCoordenadaFinEntidad(elemento);
490         Coordenadas dato2=ColeccionFunciones.ObtenerCoordenadaInicioEntidad(elemento);
491         if (elemento.orientacion == 0) {
492             linea=avanceSegmentoCircularAntihorario;
493         }else{
494             linea=avanceSegmentoCircularHorario;
495         }
496             linea=linea.replace("x",Double.toString(dato.x));
497             linea=linea.replace("y", Double.toString(dato.y));
498             linea=linea.replace("i", Double.toString((double)
    FormatoNumeros.formatearNumero(elemento.Xcentro-dato2.x)));
499             linea=linea.replace("j", Double.toString((double)
    FormatoNumeros.formatearNumero(elemento.Ycentro-dato2.y)));
500             return linea;
501     }
502
503     /**
504      * Method to generate the circular feed.
505      * @param elemento the circle to translate.
506      * @return a string.
507      */
508     public static String avanceCicular(DatosCirculo elemento) {
509         String linea1=null;
510         String linea2=null;
511         String linea=null;
512         Coordenadas dato=ColeccionFunciones.ObtenerCoordenadaFinEntidad(elemento);
```

```
513         Coordenadas dato2=ColeccionFunciones.ObtenerCoordenadaInicioEntidad(elemento);
514         if(elemento.orientacion==0){
515             linea1=avanceCircularHorario;
516             linea2=avanceCircularHorario;
517         }else{
518             linea1=avanceCircularAntihorario;
519             linea2=avanceCircularAntihorario;
520         }
521         linea1=linea1.replace("x", Double.toString(dato.x));
522         linea1=linea1.replace("y",Double.toString(dato.y));
523         linea1=linea1.replace("radio", Double.toString(elemento.Radio));
524         linea1=linea1.concat("\r\n");
525         linea2=linea2.replace("x", Double.toString(dato2.x));
526         linea2=linea2.replace("y", Double.toString(dato2.y));
527         linea2=linea2.replace("radio", Double.toString(elemento.Radio));
528         linea=linea1.concat(linea2);
529         return linea;
530     }
531 
532     /**
533      * Method to get entities info for the drilling process
534      * @param elemento a circular entity.
535      * @return a string.
536      */
537     public static String coordenadasTaladro(DatosCirculo elemento) {
538         String linea = coordenadas.replace("x",Double.toString(elemento.CentroX));
539         linea=linea.replace("y", Double.toString(elemento.CentroY));
540         return linea;
541     }
542 
543     /**
544      * Method to set the console properties.
545      * @param principal is the text area with the main program.
546      * @param consola is the console of the user's interface.
547      */
548     public static void formatearConsola(JTextArea principal,JTextPane consola) {
549         consola.setContentType("text/html");
550         String allText = principal.getText() ;
551         String formateo="";
552         StringTokenizer st = new StringTokenizer(allText,"\r\n") ;
553         formateo= formateo+"<font color=\"black\">(CODIGO DEL PROGRAMA
    PRINCIPAL</font><br>";
554         formateo= formateo+"<font color=\"black\">UBICACION: "+Ruta+nombreA+")</font><br>";
555         while (st.hasMoreTokens()) {
556              String line = st.nextToken();
557              formateo= formateo+"<font color=\"black\">"+line+"</font><br>";
558             }
559         allText=plano.getText();
560         if(allText.isEmpty()!=true){
561         st = new StringTokenizer(allText,"\r\n") ;
562         formateo= formateo+"<font color=\"#87828B\">(CODIGO DE PLANEADO)</font><br>";
563         while (st.hasMoreTokens()) {
564              String line = st.nextToken();
565              formateo= formateo+"<font color=\"#87828B\">"+line+"</font><br>";
566             }
567         }
568         allText=contorneado.getText();
569         if(allText.isEmpty()!=true){
570         st = new StringTokenizer(allText,"\r\n") ;
571          formateo= formateo+"<font color=\"blue\">(CODIGO DE CONTORNEADO)</font><br>";
572         while (st.hasMoreTokens()) {
573              String line = st.nextToken();
```

```
574             formateo= formateo+"<font color=\"blue\">"+line+"</font><br>";
575            }
576        }
577        allText=grabado.getText();
578        if(allText.isEmpty()!=true){
579        st = new StringTokenizer(allText,"\r\n") ;
580        formateo= formateo+"<font color=\"#0CB7F2\">(CODIGO DE GRABADO)</font><br>";
581        while (st.hasMoreTokens()) {
582             String line = st.nextToken();
583             formateo= formateo+"<font color=\"#0CB7F2\">"+line+"</font><br>";
584            }
585        }
586        allText=taladro.getText();
587        if(allText.isEmpty()!=true){
588        st = new StringTokenizer(allText,"\r\n") ;
589        formateo= formateo+"<font color=\"#00ff00\">(CODIGO DE TALADRADO)</font><br>";
590        while (st.hasMoreTokens()) {
591             String line = st.nextToken();
592             formateo= formateo+"<font color=\"#00ff00\">"+line+"</font><br>";
593            }
594        }
595        consola.setText(formateo);
596        }
597    }
598
599
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggGCode;
14
15 import java.util.Hashtable;
24 /**
25  * This class implements the necessary algorithms to generate the G-Code from contours.
26  * @author: Celeste G. Guagliano
27  * @version: 13/01/15
28  *
29  */
30 public class GCodeMetodoContorneado extends GCode {
31     public static double tramo=0;
32     public static double incrementoTramo=0;
33
34 /**
35  * Method to generate the contouring.
36  * @param lista is the list of entities.
37  * @param rasgo is the text area were the code is generated.
38  * @param pasada is the partial depth of the contouring.
39  * @param repeticiones is the number of times that the subprogram must be called if the code
   is generated in a subprogram.
40  * @param herramienta is the tool involved in the process.
41  */
42     public JTextArea mecanizarLista(Hashtable lista, JTextArea rasgo,
43             double pasada, int repeticiones, Herramienta herramienta) {
44             if(repeticiones>0){
45                 --repeticiones;
46                 incrementoTramo=pasada/lista.size();
47                 rasgo.append(coordenadaAbsoluta);
48                 if(tramo==0){
49                     tramo-=incrementoTramo;
50                 }
51                 for (int i = 1; i <= lista.size(); i++) {
52                     datos elemento1 = (datos) lista.get(i);
53                     String linea = elemento1.mecanizate();
54                     tramo=(double) FormatoNumeros.formatearNumero(tramo);
55                     linea= linea+rampaZ.replace("z",Double.toString(tramo));
56                     rasgo.append(linea);
57                     tramo-=incrementoTramo;
58                         }
59                 mecanizarLista(lista,rasgo,pasada,repeticiones,herramienta);
60             }
61                 else{
62                     for (int i = 1; i <= lista.size(); i++) {
63                         datos elemento1 = (datos) lista.get(i);
64                         String linea = elemento1.mecanizate();
65                         linea=linea+("\r\n");
66                         rasgo.append(linea);
67                            }
68                         tramo=0;
69                         }
70             return rasgo;
71     }
72 }
73
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggGCode;
14
15 import java.util.Hashtable;
23
24 /**
25  * This class implements the necessary algorithms to generate the G-Code from any list of
   entities.
26  * @author: Celeste G. Guagliano
27  * @version: 13/01/15
28  *
29  */
30
31 public class GCodeMetodoGeneral extends GCode {
32     String nume=null;
33     double profundidad=0;
34     String RapidoZ;
35
36     /**
37      * Method to generate the general translation of features.
38      * @param lista is the list of entities.
39      * @param rasgo is the text area were the code is generated.
40      * @param pasada is the partial depth of the contouring.
41      * @param repeticiones is the number of times that the subprogram must be called if the
   code is generated in a subprogram.
42      * @param herramienta is the tool involved in the process.
43      */
44     public JTextArea mecanizarLista(Hashtable lista, JTextArea rasgo,
45             double pasada, int repeticiones, Herramienta herramienta) {
46         profundidad=inicializarProfundidad(pasada);
47         double zsafe=herramienta.Zseguro;
48         if(repeticiones>0){
49             rasgo.append(coordenadaAbsoluta);
50             for (int i = 1; i <= lista.size(); i++) {
51                 datos elemento1 = (datos) lista.get(i - 1);
52                 datos elemento2 = (datos) lista.get(i);
53                 boolean continuidad = true;
54                 if (elemento1 != null) {
55                     continuidad = ColeccionFunciones.compartenCoordenada(
56                             elemento1, elemento2);
57                 }
58               if (continuidad == true) {
59                     String linea = elemento2.mecanizate();
60                     linea=linea+("\r\n");
61                     rasgo.append(linea);
62
63                 } else {
64                     String RapidoZ=avanceRapidoZ.replace("z", Double.toString(zsafe));
65                     rasgo.append(RapidoZ);
66                     String salto = elemento2.saltarASiguiente();
67                     rasgo.append(salto);
68                     String linea=avanceLinealZ.replace("z",Double.toString(profundidad));
69                     rasgo.append(linea);
70                     linea = elemento2.mecanizate();
71                     linea=linea+("\r\n");
72                     rasgo.append(linea);
73                 }
74             }
75             rasgo.append(RapidoZ);
76             if (repeticiones > 1) {
77                 String linea=avanceRapidoZ.replace("z", Double.toString(zsafe));
```

```
78                  rasgo.append(linea);
79                  datos elemento = (datos) lista.get(1);
80                  Coordenadas iniciales =
  ColeccionFunciones.ObtenerCoordenadaInicioEntidad(elemento);
81                  linea=avanceRapido.replace("x",Double.toString(iniciales.x));
82                  linea=linea.replace("y", Double.toString(iniciales.y));
83                  rasgo.append(linea);
84                  profundidad=profundidad-pasada;
85                  linea=avanceLinealZ.replace("z", Double.toString(profundidad));
86                  rasgo.append(linea);
87                  }
88
89              mecanizarLista(lista,rasgo,pasada,--repeticiones,herramienta);
90          }
91          return rasgo;
92      }
93
94      /**
95       * Method to get the first depth of the process
96       * @param pasada is the partial depth of the feature.
97       * @return the first depth.
98       */
99      private double inicializarProfundidad(double pasada) {
100         if(profundidad==0){
101             profundidad-=pasada;
102         }
103         return profundidad;
104     }
105
106 }
107
```

```
12
13 package cggGCode;
14
15 import java.util.Hashtable;
23
24 /**
25  * This class implements the necessary algorithms to generate the G-Code from a list of
   holes.
26  * @author: Celeste G. Guagliano
27  * @version: 13/01/15
28  *
29  */
30
31 public class GCodeMetodoTaladrado extends GCode {
32
33     /**
34      * Method to initialize the direct drilling process
35      * @param centro is the first hole's center
36      * @param profuRasgo is the total depth of the feature.
37      * @param velocidad2 is the spindle speed.
38      * @param avance2 is the tool's feed rate.
39      * @param principal is the main program's code.
40      * @param zseg is the safety height.
41      * @param zcambio is the safety tool change's height.
42      */
43     private static void inicializarTaladradoDirecto(Coordenadas centro,
44             double profuRasgo, int velocidad2, double avance2,
45             JTextArea principal, double zseg,double zcambio) {
46             String linea=taladrado.replace("referencia", Double.toString(zseg));
47             linea=linea.replace("profundidad",Double.toString(profuRasgo));
48             linea=linea.replace("planoRetraccion",Double.toString(zcambio));
49             linea=linea.replace("avance",Double.toString(avance2));
50             principal.append(linea);
51         }
52     /**
53      * Method to initialize the drilling process
54      * @param datos is the first hole's data.
55      * @param herramientas is the tool's list.
56      * @param principal is the main program code.
57      * @param profuRasgo is the total depth of the feature.
58      */
59     static void inicializarTaladrado(datos datos,
60             Hashtable herramientas, JTextArea principal, double profuRasgo) {
61         Herramienta taladro=(Herramienta) herramientas.get("Taladrado");
62         double pasada= taladro.Pasada;
63         int velocidadT=(int)taladro.Velocidad;
64         double avanceT=taladro.Avance;
65         double Zcambio=taladro.Zcambio;
66         int nroHerramienta = (int) taladro.Numero;
67         Coordenadas centro = (Coordenadas)
   ColeccionFunciones.ObtenerCoordenadaCentroEntidad(datos);
68         String cambioTool=cambioHerramienta.replaceAll("herramienta",
   Integer.toString(nroHerramienta));
69         cambioTool=cambioTool.replace("velocidad",Integer.toString(velocidadT));
70         cambioTool=cambioTool.replace("x",Double.toString(centro.x));
71         cambioTool=cambioTool.replace("y",Double.toString(centro.y));
72         cambioTool=cambioTool.replace("planoRetraccion", Double.toString(Zcambio));
73         principal.append(cambioTool);
74         double zseg=taladro.Zseguro;
75         if (pasada<profuRasgo){
76             inicializarTaladradoProfundo(centro,pasada,profuRasgo,velocidadT,avanceT,princi
```

```
   pal,zseg,Zcambio);
77         }else{
78             inicializarTaladradoDirecto(centro,profuRasgo,velocidadT,avanceT,principal,zseg,
   Zcambio);
79         }
80         }
81
82     /**
83      * Method to initialize the deep drilling process
84      * @param centro is the first hole's center.
85      * @param pasada is the partiaL depth of the feature.
86      * @param profuRasgo is the total depth of the feature.
87      * @param velocidad2 is the spindle speed.
88      * @param avance2 is the tool's feed rate.
89      * @param principal is the main's program code.
90      * @param zseg is the safety height.
91      * @param zcambio is the safety tool change's height.
92      */
93     private static void inicializarTaladradoProfundo(Coordenadas centro,
94             double pasada, double profuRasgo, int velocidad2, double avance2,
95             JTextArea principal, double zseg,double zcambio) {
96         String linea=taladradoProfundo.replace("referencia", Double.toString(zseg));
97         linea=linea.replace("profundidad",Double.toString(profuRasgo));
98         linea=linea.replace("planoRetraccion",Double.toString(zcambio));
99         linea=linea.replace("pasada",Double.toString(pasada));
100        linea=linea.replace("avance",Double.toString(avance2));
101        principal.append(linea);
102
103
104    }
105
106
107}
108
```

```java
/*
Copyright 2014, Celeste Gabriela Guagliano.

*/

package cggGCode;

import javax.swing.*;

/**
 * This class implements a directory chooser to save files
 * @author: Celeste G. Guagliano
 * @version: 13/01/15
 *
 */

public class SelectorDeDirectorio extends JPanel implements ActionListener {
    JButton go;

    static JFileChooser chooser;
    static String choosertitle;

    public static String SeleccionarDirectorio() {
        String directorio = null;

        chooser = new JFileChooser();
        chooser.setCurrentDirectory(new java.io.File("."));
        chooser.setDialogTitle(choosertitle);
        chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        chooser.setAcceptAllFileFilterUsed(false);
        if (chooser.showOpenDialog(chooser) == JFileChooser.APPROVE_OPTION) {
            System.out.println("getCurrentDirectory(): "
                    + chooser.getCurrentDirectory());
            System.out.println("getSelectedFile() : "
                    + chooser.getSelectedFile());

        } else {
            System.out.println("No Selection ");
        }
        return directorio = (chooser.getSelectedFile().getAbsolutePath() + File.separator);
    }

    public Dimension getPreferredSize() {
        return new Dimension(200, 200);
    }

    @Override
    public void actionPerformed(ActionEvent arg0) {
        // TODO Auto-generated method stub

    }

}
```

```
nombreArchivo= nombre.txt
nombreSubprograma=nombre.txt
encabezado=%\r\nOnumero\r\nG00 G90 G49 G80 G21\r\n
cambioHerramienta=Therramienta M06\r\nSvelocidad M03\r\nG00 G54 G90 Xx Yy\r\nG43 Hherramienta ZplanoRetraccion\r\n
giroHusillo=Svelocidad M03\r\n
seleccionReferencia=G00 G54 Xx Yy\r\n
compensacionHerramienta=G43 Hherramienta\r\n
definirAvance=G01 Zreferencia Favance\r\n
avanceRapidoZ=G00 Zz\r\n
avanceRapidoXY=G00 Xx Yy\r\n
avanceLineal=G01 Xx Yy
avanceLinealZ=G01 Zz\r\n
avanceLinealRampa=G01 Xx Yy Zz \r\n
rampaZ=Zz\r\n
avanceSegmentoCircularHorario=G02 Xx Yy Ii Jj
avanceSegmentoCircularAntihorario=G03 Xx Yy Ii Jj
avanceCircularHorario=G02 Xx Yy Rradio
avanceCircularAntihorario=G03 Xx Yy Rradio
taladrado=G00 ZplanoRetraccion\r\nG81 G98 Rreferencia Z-profundidad Favance\r\n
taladradoProfundo= G00 ZplanoRetraccion\r\nG83 G98 Rreferencia Z-profundidad Qpasada Favance\r\n
coordenadas=Xx Yy\r\n
finalizarCicloFijo= G80\r\n
terminacion=M30\r\n%
llamadaASubprog=M98 Pprograma Lrepeticiones\r\n
retornoDeSubprograma=M99\r\n%
subprogramas=false
coordenadaAbsoluta=G90\r\n
coordenadaRelativa=G91\r\n

```

```
nombreArchivo=%nombre
nombreSubprograma=Lnombre
encabezado=
cambioHerramienta=Therramienta M00\r\nSvelocidad M03\r\nG00 G54 G90 Xx Yy\r\nDherramienta ZplanoRetraccion\r\n
giroHusillo=Svelocidad M03\r\n
seleccionReferencia=
compensacionHerramienta=D Hherramienta\r\n
definirAvance=G01 Zreferencia Favance\r\n
avanceRapidoZ=G00 Zz\r\n
avanceRapidoXY=G00 Xx Yy\r\n
avanceLineal=G01 Xx Yy
avanceLinealZ=G01 Zz\r\n
avanceLinealRampa=G01 Xx Yy Zz\r\n
rampaZ=Zz\r\n
avanceSegmentoCircularHorario=G02 Xx Yy Ii Jj
avanceSegmentoCircularAntihorario=G03 Xx Yy Ii Jj
avanceCircularHorario=G02 Xx Yy Uradio
avanceCircularAntihorario=G03 Xx Yy Uradio
taladrado=G81 R2=referencia R3=-profundidad R10=planoRetraccion Favance\r\n
taladradoProfundo=G83 R2=referencia R3=-profundidad R5=pasada R10=planoRetraccion Favance\r\n
coordenadas=Xx Yy\r\n
finalizarCicloFijo= G80\r\n
terminacion=M30\r\n
llamadaASubprog=Lprograma Prepeticiones\r\n
retornoDeSubprograma=M17\r\n
subprogramas=true
coordenadaAbsoluta=G90\r\n
coordenadaRelativa=G91\r\n
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggOptimizacion;
14
15 import java.util.Hashtable;
18
19 /**
20  * This class implements an algorithm of optimization for lists.
21  * This kind of optimization just returns the same list that receives.
22  * @author: Celeste G. Guagliano
23  * @version: 13/01/15
24  *
25  */
26
27 public class Optimizacion {
28
29     public static Hashtable Optimizacion(Hashtable lista) {
30         Hashtable listaOptimizada = new Hashtable();
31         listaOptimizada = lista;
32         return listaOptimizada;
33     }
34
35 }
36
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggOptimizacion;
14
15 import java.util.Hashtable;
19
20 /**
21  * This class implements an algorithm of optimization for lists.
22  * This kind of optimization receives a list, sorts the elements and returns a new list.
23  * @author: Celeste G. Guagliano
24  * @version: 13/01/15
25  *
26  */
27 public class OptimizacionMetodo1 extends Optimizacion {
28
29     public static Hashtable Optimizacion(Hashtable lista) {
30         Hashtable listaOptimizada = new Hashtable();
31         listaOptimizada = ColeccionFunciones.InicializarTablaOrdenada(lista);
32         lista = ColeccionFunciones.ObtenerNuevaLista(lista, listaOptimizada);
33         listaOptimizada = ColeccionFunciones.ObtenerElementosOrdenados(
34                 listaOptimizada, lista);
35         return listaOptimizada;
36     }
37
38 }
39
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggOptimizacion;
14
15 import java.util.Hashtable;
26
27 /**
28  * This class implements an algorithm of optimization for lists.
29  * This kind of optimization receives a list, sort the items in a new list, check if they
   form a closed loop, calculates the offset entities and returns the optimized list.
30  * @author: Celeste G. Guagliano
31  * @version: 13/01/15
32  *
33  */
34 public class OptimizacionMetodo2 extends Optimizacion {
35
36     /**
37      * Method to get the optimization of an entities's list
38      * @param lista is the list of entities.
39      * @param RadioHerramienta is the tool radius.
40      * @return an optimized list.
41      */
42     public static Hashtable Optimizacion(Hashtable lista,double RadioHerramienta) {
43         Hashtable listaOptimizada = new Hashtable();
44         Hashtable listaEcuaciones= new Hashtable();
45         Hashtable listaCompensada= new Hashtable();
46         Hashtable listaEntidadesCompensadas= new Hashtable();
47         listaOptimizada = ColeccionFunciones.InicializarTablaOrdenada(lista);
48         lista = ColeccionFunciones.ObtenerNuevaLista(lista, listaOptimizada);
49         if(lista.size()>0){
50         listaOptimizada = ColeccionFunciones.ObtenerElementosOrdenados(
51                 listaOptimizada, lista);
52         }
53         boolean cerrado = ColeccionFunciones
54                 .EvaluarContornoCerrado(listaOptimizada);
55         if (cerrado == true) {
56             listaOptimizada = ColeccionFunciones.OrientarContornoCerrado(listaOptimizada);
57             listaEcuaciones=compensacionContorno.ObtenerListaEcuaciones(listaOptimizada,Rad
   ioHerramienta);
58             listaCompensada=
   compensacionContorno.intersectarEcuacionesCompensadas(listaOptimizada,listaEcuaciones);
59             return listaCompensada;
60         } else {
61             return lista;
62         }
63     }
64
65     /**
66      * Method to calculate the intersection between entities
67      * @param i is the key of the first entity.
68      * @param j is the key of the second entity.
69      * @param ecuacion1 is the equation of the first entity.
70      * @param elemento1 is the first entity.
71      * @param ecuacion2 is the equation of the second entity.
72      * @param elemento2 is the second entity.
73      * @param listaCompensada is the list of compensated entities.
74      * @return a list of entities.
75      */
76     public static Hashtable intersectarEntidades(int i,int j, EcuacionEntidad ecuacion1,
77             datos elemento1, EcuacionEntidad ecuacion2, datos elemento2, Hashtable
   listaCompensada) {
78         Coordenadas interseccion=new Coordenadas(0,0);
```

```
79          double centroY=0;
80          if(ecuacion1 instanceof EcuacionRecta){
81              if(ecuacion2 instanceof EcuacionRecta){
82                  interseccion=compensacionContorno.intersectarRectas((EcuacionRecta)ecuacion1
    ,(EcuacionRecta)ecuacion2);
83
84              }else{
85                  interseccion=compensacionContorno.intersectarRectaYArco((EcuacionRecta)ecua
    cion1,(EcuacionCircunferencia)ecuacion2,(DatosArcos)elemento2);
86
87              }
88          }else{
89              if(ecuacion2 instanceof EcuacionRecta){
90
91                  interseccion=compensacionContorno.intersectarArcoYRecta((EcuacionRecta)ecua
    cion2,(EcuacionCircunferencia)ecuacion1,(DatosArcos)elemento1);
92                  }
93              else{
94                  Coordenadas FinArco=
    ColeccionFunciones.ObtenerCoordenadaFinEntidad(elemento2);
95                  if(FinArco.y<((DatosArcos)elemento1).Ycentro){
96                      centroY=-((DatosArcos)elemento1).Ycentro;
97                      }else{
98                          centroY=((DatosArcos)elemento1).Ycentro;
99                      }
100                 interseccion=compensacionContorno.intersectarArcos((EcuacionCircunferencia)e
    cuacion1,(EcuacionCircunferencia)ecuacion2,(DatosArcos)elemento1,(DatosArcos)elemento2);
101             }
102
103         }
104         elemento1=elemento1.cambiarCoordenadaFinal(interseccion);
105         elemento2=elemento2.cambiarCoordenadaInicial(interseccion);
106         listaCompensada.put(i, elemento1);
107         listaCompensada.put(j, elemento2);
108     return listaCompensada;
109     }
110
111 }
112
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggTablas;
14
15 import java.io.File;
35
36 ;
37 /**
38  * This class access the general table of entities of DXF project and collect the admitted
   entities for DXF2Machine in a new table.
39  * @author: Celeste G. Guagliano
40  * @version: 13/01/15
41  *
42  */
43 public class Tabla {
44
45     public static HashSet ListaEntidades = new HashSet();
46     public static JTextArea principal = new JTextArea();
47     public static JTextPane consola=new JTextPane();
48
49     /**
50      * Method to get access to the original list of entities.
51      */
52     public static void AccederALaLista() {
53
54         for (int i = 0; i < myCanvas._dxf._u._myTables.size(); i++) {
55             for (int j = 0; j < myCanvas._dxf._u._myTables.elementAt(i)._myLayers
56                     .size(); j++) {
57                 for (int k = 0; k < myCanvas._dxf._u._myTables.elementAt(i)._myLayers
58                         .elementAt(j)._myEnt.size(); k++) {
59                     (myCanvas._dxf._u._myTables.elementAt(i)._myLayers
60                             .elementAt(j)._myEnt.elementAt(k)).obtenerDatos();
61
62                 }
63
64             }
65
66         }
67
68     }
69
70     /**
71      * Method to clear the list's content.
72      */
73     public static void resetearTabla() {
74         ListaEntidades.clear();
75         principal.setText("");
76
77     }
78
79     /**
80      * Method to add entities into a list.
81      * @param entidad is the entity to add.
82      */
83     public static void agregarDatoATabla(datos entidad) {
84         ListaEntidades.add(entidad);
85     }
86
87     /**
88      * Method to collect entities by a certain criteria.
89      * @param postprocesador is the machine's post processor.
90      * @param ruta is a selected directory.
```

```
 91      */
 92     public static void ObtenerTablas(String postprocesador, String ruta) {
 93
 94         Tabla.resetearTabla();
 95         Tabla.AccederALaLista();
 96         DatosCirculo centro=DatosCirculo.obtenerCentro();
 97         if (centro!=null){
 98         Hashtable ListaContorno = TablaContorno.obtenerTabla();
 99         ListaContorno=desplazarCentro(ListaContorno,centro);
100         Hashtable ListaPlaneado = TablaTocho.ObtenerTabla();
101         ListaPlaneado=desplazarCentro(ListaPlaneado,centro);
102         Hashtable ListaGrabado = TablaGrabado.ObtenerTabla();
103         ListaGrabado=desplazarCentro(ListaGrabado,centro);
104         Hashtable ListaAgujeros = TablaAgujeros.ObtenerTablaAgujereado();
105         ListaAgujeros=desplazarCentro(ListaAgujeros,centro);
106         Hashtable herramientas =
    TablaHerramientas.ObtenerTabla(DXF_Loader.TablaHerramientas);
107         double profPlano = Double.parseDouble(DXF_Loader.profPlano.getText());
108         double profConto = Double.parseDouble(DXF_Loader.profContorno.getText());
109         double profGrabo = Double.parseDouble(DXF_Loader.profGrabo.getText());
110         double profTaladro = Double.parseDouble(DXF_Loader.profTaladro.getText());
111         String postProce = (String)
    TablaPostprocesadores.ObtenerPostprocesador(postprocesador);
112         GCode.prepararPostprocesador(postProce, ruta);
113         principal = GCode.EncabezarPrograma(principal);
114         String texto=principal.getText();
115         if (DXF_Loader.plano == true) {
116             if (ListaContorno.size() != 0) {
117                 ListaPlaneado = ColeccionFunciones.ObtenerTocho(ListaContorno,
118                         herramientas);
119
120                 principal =(new GCodeMetodoGeneral()).GenerarPlaneado(new
    GCodeMetodoGeneral(),ListaPlaneado, herramientas,
121                         profPlano, principal);
122             }
123         }
124         if (DXF_Loader.contorno == true) {
125             if (ListaContorno.size() != 0) {
126                 GCode mecanizarRasgo= new GCodeMetodoContorneado();
127                 principal = mecanizarRasgo.GenerarContorneado(mecanizarRasgo,ListaContorno,
128                         herramientas, profConto, principal);
129
130             }
131         }
132         if (DXF_Loader.grabo == true) {
133             if (ListaGrabado.size() != 0) {
134                 GCode mecanizarRasgo=new GCodeMetodoGeneral();
135                 principal = mecanizarRasgo.GenerarGrabado(mecanizarRasgo,ListaGrabado,
    herramientas,
136                         profGrabo, principal);
137             }
138         }
139         if (DXF_Loader.taladro == true) {
140             if (ListaAgujeros.size() != 0) {
141                 GCode mecanizarRasgo= new GCodeMetodoTaladrado();
142                 principal = GCode.GenerarTaladrado(mecanizarRasgo,ListaAgujeros,
    herramientas,
143                         profTaladro, principal);
144
145             }
146         }
147         GCode.archivarMecanizado(principal,consola);
```

```
148         GCode.formatearConsola(principal,consola);
149     }else{
150          JOptionPane.showMessageDialog(null, "Debe seleccionar un punto de origen de pieza
    unico", "Error", JOptionPane.ERROR_MESSAGE);
151     }
152 }
153 
154     /**
155      * Method to set the reference point of a map.
156      * @param lista is the entities's list
157      * @param centro is the center of the reference point.
158      * @return a list of entities.
159      */
160     private static Hashtable desplazarCentro(Hashtable lista,
161             DatosCirculo centro) {
162         Hashtable listaDesplazada=new Hashtable();
163         for(Enumeration e=lista.elements();e.hasMoreElements();){
164             datos elemento=(datos) e.nextElement();
165             elemento=elemento.desplazateAlCentro(centro);
166             listaDesplazada.put(listaDesplazada.size()+1,elemento);
167         }
168         return listaDesplazada;
169     }
170 
171 
172 }
173 
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggTablas;
14
15 import java.util.Enumeration;
25
26 /**
27  * This class access the general table of entities of DXF2Machine and collect the entities
    matching a color code in a new table.
28  * @author: Celeste G. Guagliano
29  * @version: 13/01/15
30  *
31  */
32
33 public class TablaAgujeros {
34     public static Hashtable ListaAgujeros = new Hashtable();
35     public static int colorAgujeros = 3;
36
37     /**
38      * Method to get a hole's list.
39      * @return a hole's list.
40      */
41     public static Hashtable ObtenerTablaAgujereado() {
42         Hashtable Lista= new Hashtable();
43         resetearTablas();
44         Lista = ColeccionFunciones.ObtenerSubconjunto(
45                 Tabla.ListaEntidades, colorAgujeros);
46         ListaAgujeros=ColeccionFunciones.ObtenerCirculos(Lista);
47         return ListaAgujeros;
48     }
49
50     /**
51      * Method to reset the content of the list and the text area were the code is generated.
52      */
53     public static void resetearTablas() {
54         ListaAgujeros.clear();
55         GCode.taladro.setText(" ");
56     }
57
58
59 }
60
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggTablas;
14
15 import java.util.Enumeration;
24
25 /**
26  * This class access the general table of entities of DXF2Machine and collect the entities
   matching a color code in a new table.
27  * This kind of optimization just returns the same list that receives.
28  * @author: Celeste G. Guagliano
29  * @version: 13/01/15
30  *
31  */
32
33 public class TablaContorno {
34     public static Hashtable ListaContorno = new Hashtable();
35     public static int colorContorno = 5;
36
37     /**
38      * Method to get the entities of the contour list.
39      * @return an entities list.
40      */
41
42     public static Hashtable obtenerTabla() {
43         resetearTablas();
44         ListaContorno = ColeccionFunciones.ObtenerSubconjunto(
45         Tabla.ListaEntidades, colorContorno);
46         return ListaContorno;
47     }
48
49     /**
50      *  Method to reset the content of the list and the text area were the code is
   generated.
51      */
52     public static void resetearTablas() {
53         ListaContorno.clear();
54         GCode.contorneado.setText("");
55     }
56
57 }
58
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggTablas;
14
15 import java.util.Enumeration;
24
25 /**
26  * This class access the general table of entities of DXF2Machine and collect the entities
    matching a color code in a new table.
27  * @author: Celeste G. Guagliano
28  * @version: 13/01/15
29  *
30  */
31
32 public class TablaGrabado {
33     public static Hashtable ListaGrabado = new Hashtable();
34     public static int colorGrabado = 4;
35
36     /**
37      * Method to get the engraving list.
38      * @return an entities list.
39      */
40     public static Hashtable ObtenerTabla() {
41         resetearTablas();
42         ListaGrabado = ColeccionFunciones.ObtenerSubconjunto(
43         Tabla.ListaEntidades, colorGrabado);
44         return ListaGrabado;
45
46     }
47
48     /**
49      * Method to reset the engraving list and the text area were the code is generated.
50      */
51     public static void resetearTablas() {
52         ListaGrabado.clear();
53         GCode.grabado.setText("");
54
55     }
56
57 }
58
```

```
 2 Copyright 2014, Celeste Gabriela Guagliano.
12
13 package cggTablas;
14
15 import java.util.Hashtable;
23
24 /**
25  * This class generate a table of tools.
26  * @author: Celeste G. Guagliano
27  * @version: 13/01/15
28  *
29  */
30
31
32 public class TablaHerramientas {
33     static Hashtable TablaHerramientas = new Hashtable();
34
35     /**
36      * Method to get the tool's list
37      * @param herramientas is the tool's table.
38      * @return a tool's list.
39      */
40     public static Hashtable ObtenerTabla(JTable herramientas) {
41         for (int i = 1; i < 5; i++) {
42             double Numero = (double) herramientas.getValueAt(0, i);
43             double Diametro = (double) herramientas.getValueAt(1, i);
44             double Velocidad = (double) herramientas.getValueAt(3, i);
45             double Avance = (double) herramientas.getValueAt(2, i);
46             double Pasada = (double) herramientas.getValueAt(4, i);
47             double Zseguro= (double) herramientas.getValueAt(5,i);
48             double Zcambio=(double) herramientas.getValueAt(6, i);
49             Herramienta dato = new Herramienta(Numero, Diametro, Velocidad,
50                     Avance, Pasada,Zseguro,Zcambio);
51             TablaHerramientas.put(herramientas.getColumnName(i), dato);
52         }
53         return TablaHerramientas;
54         }
55
56      }
57
58
```

```java
/*
Copyright 2014, Celeste Gabriela Guagliano.

*/

package cggTablas;

import java.util.Hashtable;

/**
 * This class generate a table of admitted postprocessors.
 * @author: Celeste G. Guagliano
 * @version: 13/01/15
 *
 */
public class TablaPostprocesadores {

    /**
     * Method to get the post-processor's list.
     * @return a list of post-processors
     */
    public static Hashtable ObtenerTabla() {
        Hashtable postprocesadores = new Hashtable();
        postprocesadores.put("Sinumerik 810/820M",
                "cggMaquinas.GCode_Sinumerik820");
        postprocesadores.put("HAAS/FANUC", "cggMaquinas.GCode_Hass");
        return postprocesadores;
    }

    /**
     * Method to get the selected post-processor
     * @param postprocesador is the key of the post-processor.
     * @return a post-processor.
     */
    public static String ObtenerPostprocesador(String postprocesador) {
        String parametro = null;
        Hashtable maquinas = ObtenerTabla();
        parametro = (String) maquinas.get(postprocesador);
        return parametro;
    }
}

```

```java
/*
 * Copyright 2014, Celeste Gabriela Guagliano.
 */

package cggTablas;

import java.util.Enumeration;

/**
 * This class resets the stock table.
 * @author: Celeste G. Guagliano
 * @version: 13/01/15
 *
 */
public class TablaTocho {
    public static Hashtable ListaTocho = new Hashtable();

    /**
     * Method to reset the stock´s list and the text area were the code is generated.
     */
    public static void resetearTablas() {
        ListaTocho.clear();
        GCode.plano.setText("");
    }

    /**
     * Method to get the stock list.
     * @return a stock list.
     */
    public static Hashtable ObtenerTabla() {
        resetearTablas();
        return ListaTocho;
    }
}

```

```
/*-------------------------------------------------------------------------------------
Copyright 2007, Stéphan Soulard and Edouard Vanhauwaert.
Copyright 2014, Celeste Gabriela Guagliano.

This file was originally part of DXF project and then modified by
Celeste Gabriela Guagliano for DXF2Machine project.

DXF2Machine is free software: you can redistribute it and/or modify it under the terms of
the GNU General Public License as published by the Free Software Foundation, either
version 2 of the License.

DXF2Machine is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with DXF2Machine.
If not, see <http://www.gnu.org/licenses/>.
  -------------------------------------------------------------------------------------
  */

/*
 * Initials     Name
 * ------------------------------------
 * CeGu         Celeste Guagliano.
 */

/*
 * modification history (new versions first)
 * -----------------------------------------------------------
 * 20150328 v0.0.5 CeGu add itemStateChanged() method
 * 20150328 v0.0.4 CeGu add isCellEditable() method
 * 20141120 v0.0.3 CeGu improve DXF_Loader() method
 * 20141002 v0.0.2 CeGu improve DXF_Loader() method
 * 20140828 v0.0.1 CeGu fork from DXF project

*/

package myDXF;

import java.awt.BasicStroke;

/**
 * This class is the main class of the DXF2Machine project
 * @author: Stephan Soulard, Edouard Vanhauwaert
 * @version: 28/03/15 by Celeste Guagliano
 *
 */
public class DXF_Loader extends JPanel implements ActionListener {
    public static ResourceBundle res = ResourceBundle
            .getBundle("myDXF.i18n.Ressources_ES");

    public static final long serialVersionUID = 1L;
    public static JTabbedPane tabPane;
    public static String Ruta = null;
    public myToolBar _typeOutil;
    public JComboBox _comboLineType;
    public myJColorChooser _jcc;
    public JSplitPane _sp;
    public JScrollPane _dxfScrollPane;
    public JScrollPane _toolScrollPane;
    public myJTree tree;
    public myUnivers _u;
```

```
166     public myJMenu treeMenu;
167     public JFrame frame;
168     public String lastOpenDXF = "";
169     public String lastSaveDXF = "";
170     public String lastSaveAsImg = "";
171     protected boolean init = false;
172     public String postprocesador = null;
173
174     public static DefaultMutableTreeNode defaultLayer;
175     public static JTextField[] textFields;
176     public static JButton back;
177     public static JButton fwd;
178     public static TextArea logText;
179     public static myCanvas _mc;
180
181     public static boolean checkLineType = false;
182
183     public static Locale locale;
184
185     public Label txtLine = new Label();
186     public Label txtPoint = new Label();
187     public Label txtArc = new Label();
188     public Label txtCircle = new Label();
189     public Label txtDimension = new Label();
190     public Label GCodigo = new Label();
191
192     public Checkbox chkWriteLog;
193     public JLabel info;
194     public JLabel clipB;
195     public JLabel sel;
196     public JLabel coordXY;
197     public JProgressBar memoryProgress;
198     public JLabel ram;
199     public boolean proximitySelection = false;
200     public final String defClipTxtA = res.getString("defClipTxtA");
201     public final String defSelTxtA = res.getString("defSelTxtA");
202     public final String txtB = "                                                                    ";
203
204     public Checkbox proximity;
205     private Checkbox chk;
206     private Checkbox ltype;
207     private JLabel lbl_ltype;
208     private JLabel lbl_thick;
209
210     private Label lbl_mem;
211
212     private Label i18n;
213
214     private JMenuItem menuItemHelp;
215
216     private JMenuItem menuItemAbout;
217
218     public JMenu menu;
219     public JMenu menuAide;
220
221     public JButton jbClearLogs;
222     public static boolean taladro;
223     public static boolean contorno;
224     public static boolean grabo;
225     public static boolean plano;
226     public static JTable TablaHerramientas = new JTable();
227     public static JTextField profPlano = new JTextField("0.5");
```

```
228     public static JTextField profContorno = new JTextField("0.5");
229     public static JTextField profGrabo = new JTextField("0.5");
230     public static JTextField profTaladro = new JTextField("0.5");
231     public JComboBox postproce;
232     public JButton GenerarCodigo;
233     public JPanel botonDefineMecanizado;
234
235     public DXF_Loader() {
236         super();
237
238         this.frame = new JFrame("DXF2Machine");
239         this.frame.setIconImage(new ImageIcon(ClassLoader
240                 .getSystemResource("images/dxf.jpg")).getImage());
241         this.frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
242         this.frame.setContentPane(this.createContentPane());
243
244         new FileDrop(this.frame, new FileDrop.Listener() {
245             @Override
246             public void filesDropped(java.io.File[] files) {
247                 try {
248                     if (files[files.length - 1].getName().toLowerCase()
249                             .lastIndexOf(".dxf") != -1) {
250                         load(files[files.length - 1].getAbsoluteFile());
251                         lastOpenDXF = files[files.length - 1].getAbsoluteFile()
252                                 .getParent();
253
254                     }
255                 } catch (Exception e) {
256                     e.printStackTrace();
257                 }
258             }
259         });
260         this.frame.addComponentListener(new ComponentListener() {
261             @Override
262             public void componentHidden(ComponentEvent arg0) {
263             }
264
265             @Override
266             public void componentMoved(ComponentEvent arg0) {
267             }
268
269             @Override
270             public void componentResized(ComponentEvent arg0) {
271                 if (DXF_Loader.this.init)
272                     DXF_Loader.this.cadrer();
273             }
274
275             @Override
276             public void componentShown(ComponentEvent arg0) {
277             }
278         });
279
280         Dimension minimumSize = new Dimension(400, 300);
281
282         tabPane = new JTabbedPane();
283         JTabbedPane tab2Pane=new JTabbedPane();
284         JPanel globalPanel = new JPanel(new GridLayout(0, 1));
285         JPanel definirMecanizado= new JPanel(new GridLayout(0,1));
286         JPanel botonDefineMeca= new JPanel(new GridLayout(1,1));
287         JPanel toolPanel = new JPanel(new GridLayout(0, 1));
288         JPanel optionPanel = new JPanel(new GridLayout(0, 1));
289         JPanel statsPanel = new JPanel(new GridLayout(0, 1));
```

```
290
291         // OPTION
292         JPanel mecanizados = new JPanel(new GridLayout(0, 1));
293         JPanel postProcesado = new JPanel(new GridLayout(0, 1));
294         JPanel TiposyProfu = new JPanel(new GridLayout(0, 3));
295         statsPanel.add(mecanizados, BorderLayout.PAGE_START);
296         // mecanizados.setBorder(BorderFactory.createEmptyBorder(25, 15, 75,
297         // 5));
298         String postprocesadores[] = { "Sinumerik 810/820M", "HAAS/FANUC" };
299         postproce = new JComboBox(postprocesadores);
300         postprocesador = (String) postproce.getSelectedItem();
301         postProcesado.add(postproce);
302         Checkbox planeado = new Checkbox(res.getString("Planeado"));//"Planeado");
303         Checkbox contorneado = new Checkbox(res.getString("Contorneado"));
304         Checkbox grabado = new Checkbox(res.getString("Grabado"));
305         Checkbox taladrado = new Checkbox(res.getString("Taladrado"));
306
307         TiposyProfu.add(planeado);
308         TiposyProfu.add(new JLabel("Prof Planeado:"));
309         TiposyProfu.add(profPlano);
310         TiposyProfu.add(contorneado);
311         TiposyProfu.add(new JLabel("Prof Contorno:"));
312         TiposyProfu.add(profContorno);
313         TiposyProfu.add(grabado);
314         TiposyProfu.add(new JLabel("Prof Grabado:"));
315         TiposyProfu.add(profGrabo);
316         TiposyProfu.add(taladrado);
317         TiposyProfu.add(new JLabel("Prof Taladrado:"));
318         TiposyProfu.add(profTaladro);
319
320         JPanel generadorG = new JPanel(new GridLayout(0, 1));
321         GenerarCodigo = new JButton("Generar Codigo");
322         generadorG.add(GenerarCodigo);
323         statsPanel.add(mecanizados);
324         postProcesado
325                 .setBorder(BorderFactory.createEmptyBorder(15, 35, 30, 15));
326         TiposyProfu.setSize(150, 150);
327         TiposyProfu.setBorder(BorderFactory.createEmptyBorder(0, 35, 10, 15));
328         generadorG.setBorder(BorderFactory.createEmptyBorder(15, 35, 35, 15));
329         GenerarCodigo.addActionListener(new ActionListener() {
330             @Override
331             public void actionPerformed(ActionEvent e) {
332                 if (e.getSource() == GenerarCodigo) {
333                     Ruta = SelectorDeDirectorio.SeleccionarDirectorio();
334                     Tabla.ObtenerTablas(postprocesador, Ruta);
335                 }
336             }
337         });
338         postproce.addActionListener(this);
339
340         mecanizados.add(postProcesado, BorderLayout.PAGE_START);
341         mecanizados.add(TiposyProfu, BorderLayout.PAGE_START);
342         mecanizados.add(generadorG, BorderLayout.PAGE_START);
343
344         contorneado.addItemListener(new ItemListener() {
345             @Override
346             public void itemStateChanged(ItemEvent item) {
347                 int status = item.getStateChange();
348                 if (status == ItemEvent.SELECTED)
349                     contorno = true;
350                 else
351                     contorno = false;
```

```
352             }
353
354         });
355         planeado.addItemListener(new ItemListener() {
356             @Override
357             public void itemStateChanged(ItemEvent item) {
358                 int status = item.getStateChange();
359                 if (status == ItemEvent.SELECTED)
360                     plano = true;
361                 else
362                     plano = false;
363             }
364
365         });
366         grabado.addItemListener(new ItemListener() {
367             @Override
368             public void itemStateChanged(ItemEvent item) {
369                 int status = item.getStateChange();
370                 if (status == ItemEvent.SELECTED)
371                     grabo = true;
372                 else
373                     grabo = false;
374             }
375
376         });
377
378         taladrado.addItemListener(new ItemListener() {
379             @Override
380             public void itemStateChanged(ItemEvent item) {
381                 int status = item.getStateChange();
382                 if (status == ItemEvent.SELECTED)
383                     taladro = true;
384                 else
385                     taladro = false;
386             }
387
388         });
389
390         /*
391          * proximity = new Checkbox(res.getString("CHK_PROXIMITY"));
392          * proximity.addItemListener(new ItemListener(){
393          *
394          * @Override public void itemStateChanged(ItemEvent item) { int status =
395          * item.getStateChange(); if (status == ItemEvent.SELECTED)
396          * DXF_Loader.this.proximitySelection = true; else
397          * DXF_Loader.this.proximitySelection = false; }
398          *
399          * });
400          *
401          * chk = new Checkbox(res.getString("DXF_Loader.32"));
402          * chk.addItemListener(new ItemListener(){
403          *
404          * @Override public void itemStateChanged(ItemEvent item) { int status =
405          * item.getStateChange(); if (status == ItemEvent.SELECTED)
406          * myUnivers.antialiasing = true; else myUnivers.antialiasing = false; }
407          *
408          * });
409          *
410          * ltype = new Checkbox(res.getString("CHK_LINE_STYLE"));
411          * ltype.addItemListener(new ItemListener(){
412          *
413          * @Override public void itemStateChanged(ItemEvent item) { int status =
```

```
414          * item.getStateChange(); if (status == ItemEvent.SELECTED)
415          * checkLineType = true; else { checkLineType = false;
416          * myCanvas.big.setStroke(myTable.defaultStroke); } }
417          *
418          * });
419          */
420         i18n = new Label(res.getString(res.getString("DXF_Loader.9")));
421
422         String[] columnNames = { "Herramienta", "Planeado", "Contorneado",
423                 "Grabado", "Taladrado" };
424         Object[][] data = {
425                 { "Nro", (double) 1, (double) 2, (double) 3, (double) 4 },
426                 { "Diametro", (double) 25, (double) 12, (double) 4, (double) 6 },
427                 { "Velocidad", (double) 1000, (double) 2000, (double) 2000,
428                         (double) 2000 },
429                 { "Avance", (double) 400, (double) 200, (double) 200,
430                         (double) 100 },
431                 { "Pasada", (double) 0.5, (double) 0.5, (double) 0.5,
432                         (double) 0.5 },
433                 { "Z seguro",(double)5, (double)5,  (double)5,  (double)5},
434                 { "Z cambio",(double)50,(double)50,(double)50,(double)50}
435                         };
436         DefaultTableModel modelo = new DefaultTableModel(data, columnNames) {
437             @Override
438             public Class getColumnClass(int columna) {
439                 if (columna == 0)
440                     return String.class;
441                 return Double.class;
442             }
443
444             public boolean isCellEditable(int row, int column) {
445                 if (column == 0) {
446                     return false;
447                 } else {
448                     return true;
449                 }
450             }
451         };
452
453         TablaHerramientas = new JTable(modelo);
454         TablaHerramientas.setRowHeight(30);
455         TablaHerramientas.setPreferredScrollableViewportSize(new Dimension(300,
456                 300));
457         JScrollPane Scroll = new JScrollPane(TablaHerramientas);
458         optionPanel.add(Scroll);
459
460         String[] listeCAP = { res.getString("DXF_Loader.54"),
461                 res.getString("DXF_Loader.55"), res.getString("DXF_Loader.56") };
462         JComboBox comboCAP = new JComboBox(listeCAP);
463         /*
464          * comboCAP.addActionListener(new ActionListener(){
465          *
466          * @Override public void actionPerformed(ActionEvent e) { JComboBox cb =
467          * (JComboBox)e.getSource(); if(cb.getSelectedIndex() == 0)
468          * myTable.CAP=BasicStroke.CAP_ROUND; else if(cb.getSelectedIndex() ==
469          * 1) myTable.CAP=BasicStroke.CAP_BUTT; else if(cb.getSelectedIndex() ==
470          * 2) myTable.CAP=BasicStroke.CAP_SQUARE; if
471          * (e.getSource=="Generar Codigo"){
472          *
473          * } } } );
474          */
475         String[] listeJOIN = { res.getString("DXF_Loader.58"),
```

```
476                 res.getString("DXF_Loader.59"), res.getString("DXF_Loader.60") };
477         // JComboBox comboJOIN = new JComboBox(listeJOIN);
478         // comboJOIN.addActionListener(new ActionListener(){
479 
480     /*  String[] listeBGColor = { res.getString("DXF_Loader.62"),
481                 res.getString("DXF_Loader.63") };
482         JComboBox comboBGColor = new JComboBox(listeBGColor);
483         /*
484          * comboBGColor.addActionListener(new ActionListener(){
485          *
486          * @Override public void actionPerformed(ActionEvent e) { JComboBox cb =
487          * (JComboBox)e.getSource(); if(cb.getSelectedIndex() == 0)
488          * fmyUnivers._bgColor=Color.BLACK; else myUnivers._bgColor=Color.WHITE;
489          * } });
490          */
491 
492         // TOOLS
493 
494         // MOVE TOOLS
495         JPanel movePanel = new JPanel(new GridLayout(1, 10));
496         movePanel.setMinimumSize(new Dimension(90, 30));
497         movePanel.setPreferredSize(new Dimension(90, 30));
498         JToolBar moveBar = new JToolBar(res.getString("DXF_Loader.8"));
499 
500         JButton zoomp = new JButton();
501         zoomp.setActionCommand("zoom+");
502         zoomp.addActionListener(this);
503         zoomp.setToolTipText(res.getString("DXF_Loader.67"));
504         zoomp.setIcon(new ImageIcon(ClassLoader
505                 .getSystemResource("images/zoomin.gif")));
506         movePanel.add(zoomp, BorderLayout.SOUTH);
507 
508         JButton zoomm = new JButton();
509         zoomm.setActionCommand("zoom-");
510         zoomm.setToolTipText(res.getString("DXF_Loader.70"));
511         zoomm.addActionListener(this);
512         zoomm.setIcon(new ImageIcon(ClassLoader
513                 .getSystemResource("images/zoomout.gif")));
514         movePanel.add(zoomm, BorderLayout.SOUTH);
515 
516         back = new JButton();
517         back.setActionCommand("back");
518         back.setToolTipText(res.getString("DXF_Loader.73"));
519         back.addActionListener(this);
520         back.setIcon(new ImageIcon(ClassLoader
521                 .getSystemResource("images/undo.gif")));
522         back.setEnabled(false);
523         movePanel.add(back, BorderLayout.SOUTH);
524 
525         fwd = new JButton();
526         fwd.setActionCommand("fwd");
527         fwd.setToolTipText(res.getString("DXF_Loader.76"));
528         fwd.addActionListener(this);
529         fwd.setIcon(new ImageIcon(ClassLoader
530                 .getSystemResource("images/redo.gif")));
531         fwd.setEnabled(false);
532         movePanel.add(fwd);
533 
534 
535         JButton left = new JButton("");
536         left.setActionCommand("left");
537         left.setToolTipText(res.getString("DXF_Loader.80"));
```

```
538         left.setIcon(new ImageIcon(ClassLoader
539                 .getSystemResource("images/gauche.gif")));
540         left.addActionListener(this);
541         movePanel.add(left);
542 
543         JButton right = new JButton("");
544         right.setActionCommand("right");
545         right.setToolTipText(res.getString("DXF_Loader.84"));
546         right.setIcon(new ImageIcon(ClassLoader
547                 .getSystemResource("images/droite.gif")));
548         right.addActionListener(this);
549         movePanel.add(right);
550 
551         JButton up = new JButton();
552         up.setActionCommand("up");
553         up.setToolTipText(res.getString("DXF_Loader.87"));
554         up.setIcon(new ImageIcon(ClassLoader
555                 .getSystemResource("images/haut.gif")));
556         up.addActionListener(this);
557         movePanel.add(up);
558 
559         JButton down = new JButton("");
560         down.setActionCommand("down");
561         down.setToolTipText(res.getString("DXF_Loader.91"));
562         down.setIcon(new ImageIcon(ClassLoader
563                 .getSystemResource("images/bas.gif")));
564         down.addActionListener(this);
565         movePanel.add(down);
566 
567         JButton centrer = new JButton("");
568         centrer.setIcon(new ImageIcon(ClassLoader
569                 .getSystemResource("images/cadrer.jpg")));
570         centrer.setActionCommand("cadrer");
571         centrer.setToolTipText(res.getString("DXF_Loader.96"));
572         centrer.addActionListener(this);
573         movePanel.add(centrer);
574 
575         JButton btnResetSize = new JButton("");
576         btnResetSize.setIcon(new ImageIcon(ClassLoader
577                 .getSystemResource("images/reset.gif")));
578         btnResetSize.addActionListener(this);
579         btnResetSize.setToolTipText(res.getString("DXF_Loader.99"));
580         btnResetSize.setActionCommand("reset_size");
581         movePanel.add(btnResetSize);
582 
583         movePanel.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
584         moveBar.add(movePanel);
585 
586 
587         // TYPE LINE
588         JToolBar lineBar = new JToolBar(res.getString("DXF_Loader.101"));
589 
590         this._comboLineType = new JComboBox();
591         this._comboLineType.setAutoscrolls(true);
592 
593         // THICKNESS
594         SpinnerModel spinnerModel = new SpinnerNumberModel(1, 1, 256, 1);
595         JSpinner spinnerThickness = new JSpinner(spinnerModel);
596         spinnerThickness.addChangeListener(new ChangeListener() {
597             @Override
598             public void stateChanged(ChangeEvent event) {
599                 JSpinner source = (JSpinner) event.getSource();
```

```
600                 DXF_Loader.this._u.currThickness = Integer.parseInt(source
601                         .getValue().toString());
602             }
603 
604         });
605         lbl_ltype = new JLabel(res.getString("LBL_CURR_LTYPE"));
606 
607         logText = new myThreadedLogWriter();
608 
609         JPanel iLoveJavaGUI = new JPanel();
610 
611         this.chkWriteLog = new Checkbox(res.getString("CHK_LOG"));
612         this.chkWriteLog.setState(false);
613         this.chkWriteLog.addItemListener(new ItemListener() {
614             @Override
615             public void itemStateChanged(ItemEvent item) {
616                 int status = item.getStateChange();
617                 if (status == ItemEvent.SELECTED)
618                     myLog.setActiv(true);
619                 else
620                     myLog.setActiv(false);
621             }
622         });
623 
624         iLoveJavaGUI.add(this.chkWriteLog, BorderLayout.WEST);
625 
626         JLabel jl = new JLabel("                          ");
627         iLoveJavaGUI.add(jl, BorderLayout.CENTER);
628 
629         jbClearLogs = new JButton(res.getString("CLR_LOG"));
630         jbClearLogs.addActionListener(new ActionListener() {
631             @Override
632             public void actionPerformed(ActionEvent e) {
633                 logText.setText("");
634             }
635         });
636         iLoveJavaGUI.add(jbClearLogs, BorderLayout.EAST);
637 
638         // COLOR
639         this._jcc = new myJColorChooser();
640         this._jcc.setBorder(BorderFactory.createTitledBorder("Definir Rasgos
    Mecanizables"));
641         JToolBar colorBar = new JToolBar(res.getString("DXF_Loader.106"));
642         colorBar.setBounds(20,20,200,300);
643     //  _typeOutil = new myToolBar();
644     //  _typeOutil.setBounds(50,50,10,20);
645     //  botonDefineMeca.add(_typeOutil);
646         _jcc.setBounds(200,200,50,50);
647         //colorBar.add(botonDefineMeca);
648         colorBar.add(this._jcc);
649        // TREEVIEW
650         JToolBar treeBar = new JToolBar(res.getString("DXF_Loader.107"));
651 
652         // DO NOT MOVE THIS LINE
653         _mc = new myCanvas(this);
654         this.tree = new myJTree(_mc);
655         // La survie du monde en dï¿½pend !
656 
657         //JPanel treeView = new JPanel();
658         //treeView.setLayout(null);
659 
660     /*  minimumSize = new Dimension(400, 500);
```

```
661         treeBar.setMinimumSize(minimumSize);
662         treeBar.setPreferredSize(minimumSize);
663 
664     //  treeView.setPreferredSize(new Dimension(100, 100));
665 
666         treeBar.setMinimumSize(new Dimension(50, 50));
667         treeBar.setPreferredSize(new Dimension(50, 50));
668         // MISE EN FORME*/
669 
670         // toolPanel.add(this._typeOutil);
671 
672         toolPanel.add(moveBar, BorderLayout.PAGE_START);
673         moveBar.setBorder(BorderFactory.createEmptyBorder(25, 10, 50, 15));
674         //toolPanel.add(lineBar);
675         toolPanel.add(colorBar, BorderLayout.PAGE_START);
676         colorBar.setBorder(BorderFactory.createEmptyBorder(0, 0, 50, 0));
677 
678         tabPane.add("Configuraciones", toolPanel);
679         tabPane.add("Herramientas", optionPanel);
680         tabPane.add("GCode", statsPanel);
681 
682         globalPanel.add(tabPane, BorderLayout.PAGE_START);
683         tabPane.setBorder(BorderFactory.createEmptyBorder(15, 5, 50, 5));
684         definirMecanizado.setBorder(BorderFactory.createEmptyBorder(0,5, 75, 5));
685 
686 
687         //definirMecanizado.add(_typeOutil);
688         //definirMecanizado.add(_jcc);
689     /*  globalPanel.add(treeBar, BorderLayout.PAGE_END);
690         treeBar.setBorder(BorderFactory.createEmptyBorder(0, 0, 0,0));
691         treeBar.add(definirMecanizado);
692         this.add(_mc);*/
693 /*      JScrollPane main=new JScrollPane(Tabla.principal);
694         tab2Pane.add("Código Principal",main);
695         GCode.plano.setForeground(Color.MAGENTA);
696         JScrollPane plan=new JScrollPane(GCode.plano);
697         tab2Pane.add("Código Planeado",plan);
698         GCode.contorneado.setForeground(Color.BLUE);
699         JScrollPane conto=new JScrollPane(GCode.contorneado);
700         tab2Pane.add("Código Contorneado",conto);
701         GCode.grabado.setForeground(Color.CYAN);
702         JScrollPane grab=new JScrollPane(GCode.grabado);
703         tab2Pane.add("Código Grabado",grab);
704         GCode.taladro.setForeground(Color.GREEN);
705         JScrollPane tal=new JScrollPane(GCode.taladro);
706         tab2Pane.add("Código Taladrado",tal);*/
707         JScrollPane mos=new JScrollPane(Tabla.consola);
708         tab2Pane.add("Consola",mos);
709         globalPanel.add(tab2Pane);
710         this.tree._refCanva = _mc;
711         this.treeMenu = new myJMenu(_mc);
712         this._toolScrollPane = new JScrollPane(globalPanel);
713 
714         this._sp = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, globalPanel, _mc);
715         this._sp.setOneTouchExpandable(true);
716         this._sp.setDividerLocation(420);
717         this._sp.setPreferredSize(new Dimension((int) (globalPanel
718                 .getPreferredSize().getWidth() + 802), 660));
719 
720         JPanel _infoBar = new JPanel(new BorderLayout());
721         _infoBar.setSize(1, 50);
722 
```

```
723         JPanel memoryUsage = new JPanel(new BorderLayout());
724         this.ram = new JLabel(res.getString("DXF_Loader.112"));
725         this.ram.setOpaque(true);
726         this.ram.setBackground(Color.DARK_GRAY);
727         this.ram.setForeground(Color.WHITE);
728 
729         memoryUsage.add(this.ram, BorderLayout.EAST);
730         new myThreadedRam(this);
731         memoryUsage.add(this.memoryProgress, BorderLayout.CENTER);
732         memoryUsage.setPreferredSize(new Dimension(384, 10));
733         _infoBar.add(memoryUsage, BorderLayout.WEST);
734 
735         JPanel _labels = new JPanel(new BorderLayout());
736         _labels.setSize(1, 50);
737 
738         this.coordXY = new JLabel();
739         this.coordXY.setOpaque(true);
740         this.coordXY.setBackground(Color.BLACK);
741         this.coordXY.setForeground(Color.WHITE);
742         _infoBar.add(this.coordXY, BorderLayout.CENTER);
743 
744         this.clipB = new JLabel(defClipTxtA + "0" + txtB);
745         this.clipB.setOpaque(true);
746         this.clipB.setBackground(Color.BLACK);
747         this.clipB.setForeground(Color.WHITE);
748         _labels.add(this.clipB, BorderLayout.CENTER);
749 
750         this.sel = new JLabel(defSelTxtA + "0" + txtB);
751         this.sel.setOpaque(true);
752         this.sel.setBackground(Color.BLACK);
753         this.sel.setForeground(Color.WHITE);
754         _labels.add(this.sel, BorderLayout.WEST);
755 
756         this.info = new JLabel(DXF_Loader.res.getString("READY"));
757         this.info.setOpaque(true);
758         this.info.setBackground(Color.BLACK);
759         this.info.setForeground(Color.WHITE);
760         _labels.add(this.info, BorderLayout.EAST);
761 
762         _infoBar.add(_labels, BorderLayout.EAST);
763 
764         _infoBar.setOpaque(true);
765         _infoBar.setBackground(Color.BLACK);
766         _infoBar.setForeground(Color.WHITE);
767 
768         this.frame.setLocation(10, 10);
769         this.frame.setJMenuBar(this.createMenuBar());
770         this.frame.getContentPane().add(this._sp, BorderLayout.NORTH);
771         this.frame.getContentPane().add(_infoBar, BorderLayout.SOUTH);
772         this.frame.pack();
773         this.frame.setVisible(true);
774 
775         this._u = new myUnivers(new myHeader());
776         this.updateLineTypeCombo();
777         this.tree.createNodes();
778     }
779 
780     public void doInternational() {
781         proximity.setLabel(res.getString("CHK_PROXIMITY"));
782         ltype.setLabel(res.getString("CHK_LINE_STYLE"));
783         tabPane.setTitleAt(0, res.getString("TP_TOOLS"));
784         tabPane.setTitleAt(1, res.getString("TP_LOGS"));
```

```
785         tabPane.setTitleAt(2, res.getString("TP_OPTIONS"));
786         tabPane.setTitleAt(3, res.getString("TP_STATS"));
787         lbl_ltype.setText(res.getString("LBL_CURR_LTYPE"));
788         lbl_thick.setText(res.getString("LBL_CURR_THICK"));
789         chkWriteLog.setLabel(res.getString("CHK_LOG"));
790         jbClearLogs.setText(res.getString("CLR_LOG"));
791         lbl_mem.setText(res.getString("LBL_MEM"));
792         sel.setText(res.getString("defSelTxtA") + _mc.vectClickOn.size() + txtB);
793         clipB.setText(res.getString("defClipTxtA") + _mc.clipBoard.size()
794                 + txtB);
795         i18n.setText(res.getString("INTL"));
796         myJMenu.file_filter = res.getString("FILE_FILTER");
797         myJMenu.dialogNAME = DXF_Loader.res.getString("NEW_NAME");
798         info.setText(res.getString("READY"));
799         menuItemHelp.setText(res.getString("menuItemHelp"));
800         menuItemAbout.setText(res.getString("menuItemAbout"));
801 
802         menu.setText(DXF_Loader.res.getString("menuItemFichier"));
803 
804         myJMenu.menuItemNouveau.setText(DXF_Loader.res
805                 .getString("menuItemNouveau"));
806         myJMenu.menuItemOuvrir.setText(DXF_Loader.res
807                 .getString("menuItemOuvrir"));
808         myJMenu.menuItemEnregistrer.setText(DXF_Loader.res
809                 .getString("menuItemEnregistrer"));
810         myJMenu.menuItemEnregistrerSous.setText(DXF_Loader.res
811                 .getString("menuItemEnregistrerSous"));
812         myJMenu.menuItemRenommer.setText(DXF_Loader.res
813                 .getString("menuItemRenommer"));
814         myJMenu.menuItemExporter.setText(DXF_Loader.res
815                 .getString("menuItemExporter"));
816         myJMenu.menuItemImprimer.setText(DXF_Loader.res
817                 .getString("menuItemImprimer"));
818         myJMenu.menuItemQuitter.setText(DXF_Loader.res
819                 .getString("menuItemQuitter"));
820     }
821 
822     public void updateLineTypeCombo() {
823         Vector lt = this._u.getLTypes();
824         this._comboLineType.removeAllItems();
825 
826         for (int i = 0; i < lt.size(); i++)
827             this._comboLineType.addItem((lt.get(i)));
828     }
829 
830     public void updateStats() {
831         this.txtPoint.setText(String.valueOf(myStats.nbPoint));
832         this.txtLine.setText(String.valueOf(myStats.nbLine));
833         this.txtArc.setText(String.valueOf(myStats.nbArc));
834         this.txtCircle.setText(String.valueOf(myStats.nbCercle));
835         this.txtDimension.setText(String.valueOf(myStats.nbDimension));
836     }
837 
838     public JMenuBar createMenuBar() {
839 
840         // JMenuItem menuItem = null;
841         JMenuBar menuBar = new JMenuBar();
842         Vector<JMenuItem> vm = myJMenu.getFileMenuItems(_mc, false);
843 
844         menu = new JMenu(res.getString("menuItemFichier"));
845         menu.setMnemonic(KeyEvent.VK_A);
846 
```

```
847         for (int i = 0; i < vm.size(); i++) {
848             if (vm.elementAt(i).getText().equals("Separateur")) {
849                 menu.addSeparator();
850             } else {
851                 menu.add(vm.elementAt(i));
852             }
853         }
854         menuBar.add(menu);
855 
856         menuAide = new JMenu(res.getString("DXF_Loader.147"));
857 
858         menuItemHelp = new JMenuItem(res.getString("menuItemHelp"));
859         menuItemHelp.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_A,
860                 ActionEvent.CTRL_MASK));
861         menuItemHelp.setIcon(new ImageIcon(ClassLoader
862                 .getSystemResource("images/aide.png")));
863         menuItemHelp.addActionListener(new ActionListener() {
864             @Override
865             public void actionPerformed(ActionEvent arg0) {
866                 try {
867                     // TODO --> 2
868 
869                     Runtime.getRuntime().exec(
870                             "rundll32 url.dll,FileProtocolHandler "
871                                     + res.getString("URL_HELP"));
872 
873                 } catch (IOException e) {
874                     myLog.writeLog(e.getMessage());
875                 }
876             }
877         });
878         menuAide.add(menuItemHelp);
879 
880         menuAide.addSeparator();
881         menuItemAbout = new JMenuItem(res.getString("menuItemAbout"));
882         menuItemAbout.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_P,
883                 ActionEvent.CTRL_MASK));
884         menuItemAbout.setIcon(new ImageIcon(ClassLoader
885                 .getSystemResource("images/award.png")));
886         menuItemAbout.addActionListener(new ActionListener() {
887             @Override
888             public void actionPerformed(ActionEvent arg0) {
889                 JFrame frame = new JFrame(res.getString("ABOUT_VERS"));
890                 frame.setLocation(200, 200);
891                 frame.setIconImage(new ImageIcon(ClassLoader
892                         .getSystemResource("images/award.png")).getImage());
893                 frame.setMinimumSize(new Dimension(400, 150));
894                 frame.getContentPane().setLayout(new GridLayout(0, 1));
895                 JLabel labelImage = new JLabel(new ImageIcon(ClassLoader
896                         .getSystemResource("images/about.jpg")));
897                 frame.getContentPane().add(labelImage);
898                 frame.pack();
899                 frame.setVisible(true);
900             }
901         });
902         menuAide.add(menuItemAbout);
903         menuBar.add(menuAide);
904 
905         return menuBar;
906     }
907 
908     public Container createContentPane() {
```

```
909         JPanel contentPane = new JPanel(new BorderLayout());
910         JPanel menuPane = new JPanel(new BorderLayout());
911         contentPane.setOpaque(true);
912         menuPane.setVisible(true);
913         contentPane.add(menuPane);
914         return contentPane;
915     }
916 
917     public void Center(Point double1) {
918         if (double1.getX() < (myCoord.Max / 2)) {
919             if (double1.getY() < (myCoord.Max / 2)) {
920                 myCoord.decalageX += ((myCoord.Max / 2) - double1.getX());
921                 myCoord.decalageY += ((myCoord.Max / 2) - double1.getY());
922             } else if (double1.getY() > (myCoord.Max / 2)) {
923                 myCoord.decalageX += ((myCoord.Max / 2) - double1.getX());
924                 myCoord.decalageY -= (double1.getY() - (myCoord.Max / 2));
925             }
926         } else if (double1.getY() < (myCoord.Max / 2)) {
927             myCoord.decalageX -= (double1.getX() - (myCoord.Max / 2));
928             myCoord.decalageY += ((myCoord.Max / 2) - double1.getY());
929         } else if (double1.getY() > (myCoord.Max / 2)) {
930             myCoord.decalageX -= (double1.getX() - (myCoord.Max / 2));
931             myCoord.decalageY -= (double1.getY() - (myCoord.Max / 2));
932         }
933     }
934 
935     @Override
936     public void actionPerformed(ActionEvent e) {
937         if (e.getActionCommand() == "zoom+") {
938             myCoord.Ratio += myCoord.ratioStep;
939             myHistory.saveHistory(true);
940         } else if (e.getActionCommand() == "zoom-") {
941             if ((myCoord.Ratio - myCoord.ratioStep) < 1) {
942                 myCoord.ratioStep /= 2;
943                 myCoord.Ratio -= myCoord.ratioStep;
944             } else
945                 myCoord.Ratio -= myCoord.ratioStep;
946             myHistory.saveHistory(true);
947 
948         } else if (e.getActionCommand() == "back")
949             myHistory.backToThePast();
950         else if (e.getActionCommand() == "fwd")
951             myHistory.backToTheFuture();
952         else if (e.getActionCommand() == "left") {
953             myCoord.decalageX -= 10;
954             myHistory.saveHistory(true);
955         } else if (e.getActionCommand() == "right") {
956             myCoord.decalageX += 10;
957             myHistory.saveHistory(true);
958         } else if (e.getActionCommand() == "up") {
959             myCoord.decalageY -= 10;
960             myHistory.saveHistory(true);
961         } else if (e.getActionCommand() == "down") {
962             myCoord.decalageY += 10;
963             myHistory.saveHistory(true);
964         } else if (e.getActionCommand() == "cadrer")
965             this.cadrer();
966         else if (e.getActionCommand() == "reset_size") {
967             myCoord.reset();
968             myHistory.saveHistory(true);
969         }
970         this._u._header.setLIM(_mc.getWidth(), _mc.getHeight());
```

```
971
972         this.tree.updateEnv();
973         _mc.repaint();
974         if (e.getSource() == postproce) {
975             postprocesador = (String) postproce.getSelectedItem();
976         }
977         if (e.getSource() == GenerarCodigo) {
978             Tabla.ObtenerTablas(postprocesador, Ruta);
979         }
980     }
981
982     private static void createAndShowGUI() {
983         new DXF_Loader();
984     }
985
986     public void newDXF() {
987
988         if (_mc != null) {
989             myCanvas._dxf.tree.createNodes();
990             myCoord.reset();
991             myHistory.resetHistory();
992             myStats.reset();
993             myCanvas._dxf.tree.updateEnv();
994
995             _mc.selecting = false;
996             _mc.moving = false;
997             _mc.zooming = false;
998             _mc.drawingCircle = false;
999             _mc.drawingPolyLineStart = false;
1000            _mc.drawingPolyLineEnd = false;
1001            _mc.drawingLwPolyLineStart = false;
1002            _mc.drawingLwPolyLineEnd = false;
1003            _mc.drawingArc = false;
1004            _mc.drawingArcAngleStart = false;
1005            _mc.drawingArcAngleEnd = false;
1006            _mc.drawingEllipse = false;
1007            _mc.drawingTrace = false;
1008            _mc.drawingTxt = false;
1009            _mc.drawingSolid = false;
1010        }
1011
1012        if (this._u != null)
1013            this._u.reset();
1014
1015    }
1016
1017    public void write(String nomFichier) throws Exception {
1018        FileWriter out = null;
1019        File f = null;
1020
1021        try {
1022            f = new File(nomFichier);
1023            out = new FileWriter(f);
1024
1025            if (this._u._header == null)
1026                this._u._header = new myHeader();
1027
1028            this._u.writeHeader(out);
1029            this._u.writeTables(out);
1030            this._u.writeBlocks(out);
1031            this._u.writeEntities(out);
1032
```

```
1033            out.write("EOF\n");
1034            out.close();
1035        } catch (IOException e) {
1036            myLog.writeLog(res.getString("DXF_Loader.168") + e.getMessage());
1037        }
1038        myLog.writeLog(res.getString("DXF_Loader.169"));
1039
1040    }
1041
1042    public void load(File Fichier) throws IOException, Exception {
1043        myLog.writeLog(" <------ START " + Fichier.getName() + " ------>");
1044        new myThreadedLoad(this, Fichier);
1045    }
1046
1047    public void cadrer() {
1048        myCoord.Max = this._u.getMaxSpan();
1049        int w = DXF_Loader._mc.getWidth() - 10;
1050        int h = DXF_Loader._mc.getHeight() - 10;
1051
1052        if (w < h)
1053            myCoord.Ratio = (w) / myCoord.Max;
1054        else
1055            myCoord.Ratio = (h) / myCoord.Max;
1056        myCoord.decalageX -= myCoord.dxfToJava_X(_u.lastView.getMinX())
1057                - myCoord.javaToDXF_X(0) - _u.strayX;
1058        myCoord.decalageY -= myCoord.dxfToJava_Y(_u.lastView.getMaxY())
1059                - _u.strayY;
1060
1061        myHistory.saveHistory(true);
1062        this.tree.updateEnv();
1063    }
1064
1065    public static void main(final String[] args) {
1066
1067        javax.swing.SwingUtilities.invokeLater(new Runnable() {
1068            @Override
1069            public void run() {
1070                JPopupMenu.setDefaultLightWeightPopupEnabled(false);
1071                createAndShowGUI();
1072                if ((args.length != 0)
1073                        && (args[0].toLowerCase().lastIndexOf(".dxf") != -1)) {
1074                    File f = new File(args[0]);
1075                    if (f.exists() && f.canRead()) {
1076                        try {
1077                            myCanvas._dxf.load(f);
1078                        } catch (Exception e) {
1079                            e.printStackTrace();
1080                        }
1081                        ;
1082                    }
1083                }
1084            }
1085        });
1086
1087    }
1088
1089    public boolean isCellEditable(int i, int j) {
1090        if (j == 1) {
1091            return false;
1092        }
1093        return true;
1094
```

```
1095     }
1096
1097     public void itemStateChanged(ItemEvent e) {
1098         if (e.getSource() == postproce) {
1099             postprocesador = (String) postproce.getSelectedItem();
1100
1101         }
1102     }
1103
1104 }
1105
```

```
/*-------------------------------------------------------------------------------------
Copyright 2007, Stephan Soulard and Edouard Vanhauwaert.
Copyright 2014, Celeste Gabriela Guagliano.

This file was originally part of DXF project and then modified by
Celeste Gabriela Guagliano for DXF2Machine project.

DXF2Machine is free software: you can redistribute it and/or modify it under the terms of
the GNU General Public License as published by the Free Software Foundation, either
version 2 of the License.

DXF2Machine is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with DXF2Machine.
If not, see <http://www.gnu.org/licenses/>.
  -------------------------------------------------------------------------------------*/

/*
 * Initials     Name
 * ------------------------------------
 * CeGu         Celeste Guagliano.
 */

/*
 * modification history (new versions first)
 * -----------------------------------------------------------
 * 20141002 v0.0.3 CeGu add obtenerDatos() method
 * 20141002 v0.0.2 CeGu improve draw() method
 * 20140828 v0.0.1 CeGu fork from DXF project

*/

package myDXF.Entities;

import java.awt.Graphics;

/**
 * This class to manage the arc's data
 * @author: Stephan Soulard, Edouard Vanhauwaert
 * @version: 13/01/15 by Celeste Guagliano
 *
 */
public class myArc extends myEntity {

    private static final long serialVersionUID = 1L;
    public myPoint _point = new myPoint();
    public double _radius = 0;
    protected double _angle1 = 0;
    protected double _angle2 = 0;
    protected double _2angle2 = 0;
    public double _xang1 = 0;
    public double _yang1 = 0;
    public double _xang2 = 0;
    public double _yang2 = 0;

    private Arc2D.Double _a = new Arc2D.Double();


    public myArc(double a1, double a2, myPoint p, double r,
            myLineType lineType, int c, myLayer l, int visibility,
```

```
86              double thickness) {
87          super(c, l, visibility, lineType, thickness);
88          _point = p;
89          _radius = r;
90          _angle1 = a1;
91          _angle2 = a2;
92          _thickness = thickness;
93          _color = c;
94
95          /*
96           * datosArco datos = new datosArco(myCoord.dxfToJava_X(_point.X()),
97           * myCoord.dxfToJava_Y(_point.Y()), r, a1, a2);
98           * myDXF.Entities.TablaArcos.Tabla(datos);
99           */
100         myStats.nbArc += 1;
101     }
102
103     @Override
104     public void obtenerDatos() {
105         _xang1 = _radius * Math.cos(_angle1 * Math.PI / 180) + _point.X();
106         _yang1 = _radius * Math.sin(_angle1 * Math.PI / 180) + _point.Y();
107         _xang2 = _radius * Math.cos(_angle2 * Math.PI / 180) + _point.X();
108         _yang2 = _radius * Math.sin(_angle2 * Math.PI / 180) + _point.Y();
109         _xang1=(double) FormatoNumeros.formatearNumero(_xang1);
110         _xang2=(double) FormatoNumeros.formatearNumero(_xang2);
111         _yang1=(double) FormatoNumeros.formatearNumero(_yang1);
112         _yang2=(double) FormatoNumeros.formatearNumero(_yang2);
113         double CentroX=(double) FormatoNumeros.formatearNumero(_point.X());
114         double CentroY=(double) FormatoNumeros.formatearNumero(_point.Y());
115         double Radio=(double) FormatoNumeros.formatearNumero(Math.abs(_radius));
116         double ang1=(double) FormatoNumeros.formatearNumero(_angle1);
117         double ang2=(double) FormatoNumeros.formatearNumero(_angle2);
118         datos datos = new DatosArcos(_xang1,_yang1,_xang2,_yang2, CentroX,CentroY,Radio,
    ang1,ang2, _color, false, 0, 0);
119         Tabla.agregarDatoATabla(datos);
120
121     }
122
123     public myArc() {
124         super(-1, null, 0, null, myTable.defaultThickness);
125         _point = new myPoint();
126         _radius = 0;
127
128         myStats.nbArc += 1;
129     }
130
131     public myArc(myArc orig) {
132         super(orig._color, orig._refLayer, 0, orig._lineType, orig._thickness);
133         _point = new myPoint(orig._point);
134         _radius = orig._radius;
135         _angle1 = orig._angle1;
136         _angle2 = orig._angle2;
137     }
138
139     @Override
140     public void draw(Graphics g) {
141         Graphics2D g2d = (Graphics2D) g;
142
143         super.draw(g);
144         if (_angle2 < _angle1) {
145             _2angle2 = _angle2 + 360 - _angle1;
146         } else {
```

```
147             _2angle2 = _angle2 - _angle1;
148         }
149         _a.setArcByCenter(myCoord.dxfToJava_X(_point.X()),
150                 myCoord.dxfToJava_Y(_point.Y()), _radius * myCoord.Ratio,
151                 (_angle1), _2angle2, Arc2D.OPEN);
152 
153         g2d.draw(_a);
154     }
155 
156     public static myArc read(myBufferedReader br, myUnivers univers)
157             throws NumberFormatException, IOException {
158 
159         String ligne, ligne_temp = "";
160         double x = 0, y = 0, r = 0, a1 = 0, a2 = 0, thickness = 0;
161         int visibility = 0, c = 0;
162         myLineType lineType = null;
163         myLayer l = null;
164         myLog.writeLog("> new myArc");
165 
166         while ((ligne = br.readLine()) != null && !ligne.equalsIgnoreCase("0")) {
167             ligne_temp = ligne;
168             ligne = br.readLine();
169 
170             if (ligne_temp.equalsIgnoreCase("8")) {
171                 l = univers.findLayer(ligne);
172             } else if (ligne_temp.equalsIgnoreCase("62")) {
173                 c = Integer.parseInt(ligne);
174             } else if (ligne_temp.equalsIgnoreCase("6")) {
175                 lineType = univers.findLType(ligne);
176             } else if (ligne_temp.equalsIgnoreCase("40")) {
177                 r = Double.parseDouble(ligne);
178             } else if (ligne_temp.equalsIgnoreCase("10")) {
179                 x = Double.parseDouble(ligne);
180             } else if (ligne_temp.equalsIgnoreCase("20")) {
181                 y = Double.parseDouble(ligne);
182             } else if (ligne_temp.equalsIgnoreCase("50")) {
183                 a1 = Double.parseDouble(ligne);
184             } else if (ligne_temp.equalsIgnoreCase("51")) {
185                 a2 = Double.parseDouble(ligne);
186             } else if (ligne_temp.equalsIgnoreCase("60")) {
187                 visibility = Integer.parseInt(ligne);
188             } else if (ligne_temp.equalsIgnoreCase("39")) {
189                 thickness = Double.parseDouble(ligne);
190             }
191         }
192         return new myArc(a1, a2, new myPoint(x, y, c, null, visibility, 1), r,
193                 lineType, c, l, visibility, thickness);
194     }
195 
196     @Override
197     public void write(FileWriter out) throws IOException {
198         out.write("ARC\n");
199         super.writeCommon(out);
200         out.write("10\n");
201         out.write(_point.X() + "\n");
202         out.write("20\n");
203         out.write(_point.Y() + "\n");
204         out.write("40\n");
205         out.write(_radius + "\n");
206         out.write("50\n");
207         out.write(_angle1 + "\n");
208         out.write("51\n");
```

```
209         out.write(_angle2 + "\n");
210         out.write("0\n");
211     }
212 
213     @Override
214     public String toString() {
215         return DXF_Loader.res.getString("myArc.0");
216     }
217 
218     @Override
219     public DefaultMutableTreeNode getNode() {
220         DefaultMutableTreeNode root = new DefaultMutableTreeNode(this);
221         root.add(new DefaultMutableTreeNode(new myLabel(myLabel.X, String
222                 .valueOf(_point.X()))));
223         root.add(new DefaultMutableTreeNode(new myLabel(myLabel.Y, String
224                 .valueOf(_point.Y()))));
225         root.add(new DefaultMutableTreeNode(new myLabel(myLabel.ANGLE1, String
226                 .valueOf(_angle1))));
227         root.add(new DefaultMutableTreeNode(new myLabel(myLabel.ANGLE2, String
228                 .valueOf(_angle2))));
229         root.add(new DefaultMutableTreeNode(new myLabel(myLabel.RADIUS, String
230                 .valueOf(_radius))));
231         root.add(new DefaultMutableTreeNode(new myLabel(myLabel.THICKNESS,
232                 String.valueOf(_thickness))));
233         // System.out.println(root);
234         Vector<DefaultMutableTreeNode> v = super.getCommonNode();
235         for (int i = 0; i < v.size(); i++)
236             root.add(v.get(i));
237 
238         return root;
239     }
240 
241     @Override
242     public myLabel getNewLabel(String code, Object newValue)
243             throws NumberFormatException {
244         myLabel l = null;
245 
246         if (code.equals(myLabel.ANGLE1)) {
247             _angle1 = Double.parseDouble(newValue.toString());
248             l = new myLabel(myLabel.ANGLE1, newValue.toString());
249         } else if (code.equals(myLabel.ANGLE2)) {
250             _angle2 = Double.parseDouble(newValue.toString());
251             l = new myLabel(myLabel.ANGLE2, newValue.toString());
252         } else {
253             l = getNewArcLabel(code, newValue);
254         }
255 
256         return l;
257     }
258 
259     public myLabel getNewArcLabel(String code, Object newValue)
260             throws NumberFormatException {
261         myLabel l = null;
262 
263         if (code.equals(myLabel.X)) {
264             _point.setX(Double.parseDouble(newValue.toString()));
265             l = new myLabel(myLabel.X, newValue.toString());
266         } else if (code.equals(myLabel.Y)) {
267             _point.setY(Double.parseDouble(newValue.toString()));
268             l = new myLabel(myLabel.Y, newValue.toString());
269         } else if (code.equals(myLabel.RADIUS)) {
270             _radius = Double.parseDouble(newValue.toString());
```

```
271             l = new myLabel(myLabel.RADIUS, newValue.toString());
272         } else if (code.equals(myLabel.THICKNESS)) {
273             _thickness = Double.parseDouble(newValue.toString());
274             l = new myLabel(myLabel.THICKNESS, newValue.toString());
275         } else {
276             l = super.getCommonLabel(code, newValue);
277         }
278 
279         return l;
280     }
281 
282     @Override
283     public Arc2D.Double getSelectedEntity() {
284         return _a;
285     }
286 
287     @Override
288     public void translate(double x, double y) {
289         this._point._point.x -= myCoord.getTransalation(x);
290         this._point._point.y += myCoord.getTransalation(y);
291     }
292 
293     @Override
294     public double getMinX(double min) {
295         if ((_point.X() - _radius) < min)
296             return _point.X();
297         return min;
298     }
299 
300     @Override
301     public double getMaxX(double max) {
302         if ((_point.X() + _radius) > max)
303             return _point.X();
304         return max;
305     }
306 
307     @Override
308     public double getMinY(double min) {
309         if ((_point.Y() - _radius) < min)
310             return _point.Y();
311         return min;
312     }
313 
314     @Override
315     public double getMaxY(double max) {
316         if ((_point.Y() + _radius) > max)
317             return _point.Y();
318         return max;
319     }
320 }
321 
```

```
 1 /*-------------------------------------------------------------------------------------
 2 Copyright 2007, Stephan Soulard and Edouard Vanhauwaert.
 3 Copyright 2014, Celeste Gabriela Guagliano.
 4
 5 This file was originaly part of DXF project and then modified by
 6 Celeste Gabriela Guagliano for DXF2Machine project.
 7
 8 DXF2Machine is free software: you can redistribute it and/or modify it under the terms of
 9 the GNU General Public License as published by the Free Software Foundation, either
10 version 2 of the License.
11
12 DXF2Machine is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
13 without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
14 See the GNU General Public License for more details.
15
16 You should have received a copy of the GNU General Public License along with DXF2Machine.
17 If not, see <http://www.gnu.org/licenses/>.
18   -------------------------------------------------------------------------------------*/
19
20 /*
21  * Initials     Name
22  * ------------------------------------
23  * CeGu         Celeste Guagliano.
24  */
25
26 /*
27  * modification history (new versions first)
28  * -----------------------------------------------------------
29  * 20141028 v0.0.2 CeGu add obtenerDatos() method
30  * 20150428 v0.0.1 CeGu fork from DXF project
31
32 */
33
34 package myDXF.Entities;
35
36 import java.awt.Graphics;
60
61 /**
62  * Class to manage the circle's data
63  * @author: Stephan Soulard, Edouard Vanhauwaert
64  * @version: 13/01/15 by Celeste Guagliano
65  *
66  */
67 public class myCircle extends myEntity {
68
69     private static final long serialVersionUID = 1L;
70     private Ellipse2D.Double _e = new Ellipse2D.Double();
71     public myPoint _point = new myPoint();
72     public double _radius = 0;
73     public double xinicial = 0;
74     public double yinicial = 0;
75     public double xfinal = 0;
76     public double yfinal = 0;
77     public double angI = 0;
78     public double angF = 0;
79
80
81     public myCircle(myPoint p, double r, myLineType lineType, int c, myLayer l,
82             int visibility, double thickness) {
83         super(c, l, visibility, lineType, thickness);
84         _point = p;
85         _radius = r;
```

```
 86         _color = c;
 87 
 88         myStats.nbCercle += 1;
 89     }
 90 
 91     @Override
 92     public void obtenerDatos() {
 93 
 94         xinicial = _point.X() - _radius;
 95         yinicial = _point.Y();
 96         xfinal = xinicial + 2 * _radius;
 97         yfinal = yinicial;
 98         double centroX = _point.X();
 99         double centroY = _point.Y();
100         boolean ubicado = false;
101         int posicion = 0;
102         int orientacion = 0;
103         xinicial=(double) FormatoNumeros.formatearNumero(xinicial);
104         yinicial=(double) FormatoNumeros.formatearNumero(yinicial);
105         xfinal=(double) FormatoNumeros.formatearNumero(xfinal);
106         yfinal=(double) FormatoNumeros.formatearNumero(yfinal);
107         centroX=(double) FormatoNumeros.formatearNumero(centroX);
108         centroY=(double) FormatoNumeros.formatearNumero(centroY);
109         double Radio=(double) FormatoNumeros.formatearNumero(_radius);
110         datos datos = new DatosCirculo(xinicial,yinicial,xfinal,yfinal,centroX,centroY,
    Radio, _color, ubicado, posicion, orientacion);
111         Tabla.agregarDatoATabla(datos);
112     }
113 
114     public myCircle() {
115         super(0, null, 0, null, myTable.defaultThickness);
116 
117         myStats.nbCercle += 1;
118     }
119 
120     public myCircle(myCircle orig) {
121         super(orig._color, orig._refLayer, 0, orig._lineType, orig._thickness);
122         _point = new myPoint(orig._point);
123         _radius = orig._radius;
124 
125     }
126 
127     @Override
128     public void draw(Graphics g) {
129         double x = myCoord.dxfToJava_X(_point.X());
130         double y = myCoord.dxfToJava_Y(_point.Y());
131 
132         super.draw(g);
133 
134         _e.setFrameFromCenter(x, y, (x - (_radius * myCoord.Ratio)),
135                 (y - (_radius * myCoord.Ratio)));
136         /*
137          * =new Ellipse2D.Double(myCoord.dxfToJava_X(_point.X()-_radius),
138          * myCoord.dxfToJava_Y(_point.Y()+_radius), (_radius*2*myCoord.Ratio),
139          * (_radius*2*myCoord.Ratio) );
140          */
141         ((Graphics2D) g).draw(_e);
142     }
143 
144     public static myCircle read(myBufferedReader br, myUnivers univers)
145             throws NumberFormatException, IOException {
146 
```

```
147         String ligne, ligne_temp;
148         int visibility = 0, color = 0;
149         double x = 0, y = 0, r = 0, thickness = 1;
150         myLayer l = null;
151         myLineType lineType = null;
152 
153         myLog.writeLog("> new myCircle");
154         while ((ligne = br.readLine()) != null && !ligne.equalsIgnoreCase("0")) {
155             ligne_temp = ligne;
156             ligne = br.readLine();
157 
158             if (ligne_temp.equalsIgnoreCase("8")) {
159                 l = univers.findLayer(ligne);
160             } else if (ligne_temp.equalsIgnoreCase("60")) {
161                 visibility = Integer.parseInt(ligne);
162             } else if (ligne_temp.equalsIgnoreCase("62")) {
163                 color = Integer.parseInt(ligne);
164             } else if (ligne_temp.equalsIgnoreCase("6")) {
165                 lineType = univers.findLType(ligne);
166             } else if (ligne_temp.equalsIgnoreCase("40")) {
167                 r = Double.parseDouble(ligne);
168             } else if (ligne_temp.equalsIgnoreCase("10")) {
169                 x = Double.parseDouble(ligne);
170             } else if (ligne_temp.equalsIgnoreCase("39")) {
171                 thickness = Double.parseDouble(ligne);
172             } else if (ligne_temp.equalsIgnoreCase("20")) {
173                 y = Double.parseDouble(ligne);
174             } else {
175                 myLog.writeLog("Unknown :" + ligne_temp + "(" + ligne + ")");
176             }
177         }
178         return new myCircle(new myPoint(x, y, color, l, visibility, 1), r,
179                 lineType, color, l, visibility, thickness);
180     }
181 
182     @Override
183     public void write(FileWriter out) throws IOException {
184         out.write("CIRCLE\n");
185         super.writeCommon(out);
186         out.write("40\n");
187         out.write((_radius) + "\n");
188         out.write("10\n");
189         out.write(_point.X() + "\n");
190         out.write("20\n");
191         out.write(_point.Y() + "\n");
192         out.write("39\n");
193         out.write(_thickness + "\n");
194         out.write("0\n");
195     }
196 
197     @Override
198     public String toString() {
199         return DXF_Loader.res.getString("myCircle.0");
200     }
201 
202     @Override
203     public DefaultMutableTreeNode getNode() {
204         DefaultMutableTreeNode root = new DefaultMutableTreeNode(this);
205         root.add(new DefaultMutableTreeNode(new myLabel(myLabel.X, String
206                 .valueOf(_point.X()))));
207         root.add(new DefaultMutableTreeNode(new myLabel(myLabel.Y, String
208                 .valueOf(_point.Y()))));
```

```
209         root.add(new DefaultMutableTreeNode(new myLabel(myLabel.RADIUS, String
210                 .valueOf(_radius))));
211         root.add(new DefaultMutableTreeNode(new myLabel(myLabel.THICKNESS,
212                 String.valueOf(_thickness))));
213 
214         Vector<DefaultMutableTreeNode> v = super.getCommonNode();
215         for (int i = 0; i < v.size(); i++)
216             root.add(v.get(i));
217         // System.out.println(myLabel.X);
218         // System.out.println(root);
219         // System.out.println(String.valueOf(_point.X()));
220         return root;
221     }
222 
223     @Override
224     public myLabel getNewLabel(String code, Object newValue)
225             throws NumberFormatException {
226         myLabel l = null;
227 
228         if (code.equals(myLabel.X)) {
229             _point.setX(Double.parseDouble(newValue.toString()));
230             l = new myLabel(myLabel.X, newValue.toString());
231 
232         } else if (code.equals(myLabel.Y)) {
233             _point.setY(Double.parseDouble(newValue.toString()));
234             l = new myLabel(myLabel.Y, newValue.toString());
235         } else if (code.equals(myLabel.RADIUS)) {
236             _radius = Double.parseDouble(newValue.toString());
237             l = new myLabel(myLabel.RADIUS, newValue.toString());
238         } else if (code.equals(myLabel.THICKNESS)) {
239             _thickness = Double.parseDouble(newValue.toString());
240             l = new myLabel(myLabel.THICKNESS, newValue.toString());
241         } else {
242             l = super.getCommonLabel(code, newValue);
243         }
244 
245         return l;
246     }
247 
248     @Override
249     public Ellipse2D.Double getSelectedEntity() {
250         return _e;
251     }
252 
253     @Override
254     public void translate(double x, double y) {
255         this._point._point.x -= myCoord.getTransalation(x);
256         this._point._point.y += myCoord.getTransalation(y);
257     }
258 
259     @Override
260     public double getMinX(double min) {
261         if ((_point.X() - _radius) < min)
262             return _point.X() - _radius;
263         return min;
264     }
265 
266     @Override
267     public double getMaxX(double max) {
268         if ((_point.X() + _radius) > max)
269             return _point.X() + _radius;
270         return max;
```

```
271     }
272
273     @Override
274     public double getMinY(double min) {
275         if ((_point.Y() - _radius) < min)
276             return _point.Y() - _radius;
277         return min;
278     }
279
280     @Override
281     public double getMaxY(double max) {
282         if ((_point.Y() + _radius) > max)
283             return _point.Y() + _radius;
284         return max;
285     }
286
287     public static Ellipse2D.Double getSmallerGraphicEntity(Ellipse2D.Double orig) {
288         Ellipse2D.Double ret = new Ellipse2D.Double();
289         ret.setFrameFromCenter(orig.getCenterX(), orig.getCenterY(),
290                 orig.getMaxX() - 10, orig.getMaxY() - 10);
291         return ret;
292     }
293
294 }
295
```

```
/*-------------------------------------------------------------------------------------
Copyright 2007, Stephan Soulard and Edouard Vanhauwaert.
Copyright 2014, Celeste Gabriela Guagliano.

This file was originaly part of DXF project and then modified by
Celeste Gabriela Guagliano for DXF2Machine project.

DXF2Machine is free software: you can redistribute it and/or modify it under the terms of
the GNU General Public License as published by the Free Software Foundation, either
version 2 of the License.

DXF2Machine is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with DXF2Machine.
If not, see <http://www.gnu.org/licenses/>.
  -------------------------------------------------------------------------------------
  */

/*
 * Initials     Name
 * ------------------------------------
 * CeGu         Celeste Guagliano.
 */

/*
 * modification history (new versions first)
 * -----------------------------------------------------------
 * 20141002 v0.0.2 CeGu add obtenerDatos() method
 * 20140428 v0.0.1 CeGu fork from DXF project

*/

/**
 * Class to manage the entities's data.
 * @author: Stephan Soulard, Edouard Vanhauwaert
 * @version: 13/01/15 by Celeste Guagliano
 *
 */
package myDXF.Entities;

import java.awt.BasicStroke;

public abstract class myEntity {

    private static final long serialVersionUID = 1L;

    public myLineType _lineType;
    public int _color;
    public myLayer _refLayer;
    public double _thickness;

    public boolean isVisible = true;
    public boolean selected = false;
    public boolean changing = false;

    public abstract myLabel getNewLabel(String code, Object newValue)
            throws NumberFormatException;

    public abstract DefaultMutableTreeNode getNode();

```

```
80      public abstract void write(FileWriter out) throws IOException;
81
82      public abstract Object getSelectedEntity();
83
84      public abstract double getMinX(double min);
85
86      public abstract double getMaxX(double min);
87
88      public abstract double getMinY(double min);
89
90      public abstract double getMaxY(double min);
91
92      public abstract void translate(double x, double y);
93
94      public BasicStroke _stroke;
95
96      public myEntity(int c, myLayer l, int visibility, myLineType lineType,
97              double thickness) {
98          _lineType = lineType;
99          _refLayer = l;
100         _color = c;
101         _thickness = thickness;
102
103         if (visibility == 0)
104             isVisible = true;
105         else
106             isVisible = false;
107
108         if (_lineType != null)
109             _stroke = new BasicStroke((float) _thickness, myTable.CAP,
110                     myTable.JOIN, 10.0f, myLineType.parseTxt(_lineType._value),
111                     0.0f);
112         else
113             _stroke = new BasicStroke((float) _thickness, myTable.CAP,
114                     myTable.JOIN, 10.0f, myTable.defautMotif, 0.0f);
115
116         myCanvas._dxf.updateStats();
117     }
118
119     public myLabel getCommonLabel(String code, Object newValue)
120             throws NumberFormatException {
121         myLabel l = null;
122         if (code.equals(myLabel.COLOR)) {
123             _color = Integer.parseInt(newValue.toString());
124             l = new myLabel(myLabel.COLOR, newValue.toString());
125         } else if (code.equals(myLabel.VISISBILITY)) {
126             if (Integer.parseInt(newValue.toString()) == 0)
127                 isVisible = true;
128             else
129                 isVisible = false;
130             l = new myLabel(myLabel.VISISBILITY, newValue.toString());
131         } else if (code.equals(myLabel.TYPE_LIGNE)) {
132             // _lineType = myLineType.getStrokeID(newValue.toString());//find
133             l = new myLabel(myLabel.TYPE_LIGNE, newValue.toString());
134         }
135         return l;
136     }
137
138     public void draw(Graphics g) {
139
140         BasicStroke _lineStroke = null;
141         int c = _color;
```

```
142
143         if (this.changing)
144             c = DXF_Color.getColor(DXF_Color.getChangingColor());
145         else {
146             if (!(this instanceof myBlockReference)
147                     && !(this instanceof myLayer)) {
148                 if ((c < 0) || (_color == 255 && _refLayer != null))
149                     if (_refLayer == null)
150                         c = DXF_Color.getDefaultColorIndex();
151                     else
152                         c = _refLayer._color;
153             }
154         }
155
156         if (c < 0)
157             c = DXF_Color.getDefaultColorIndex();
158
159         // Gestion de la couleur en fonction de la couleur de fond d'écran
160         try {
161             if (DXF_Color.getColor(c).equals(myUnivers._bgColor)) {
162                 if (myUnivers._bgColor.equals(Color.WHITE))
163                     g.setColor(Color.BLACK);
164                 else if (myUnivers._bgColor.equals(Color.BLACK))
165                     g.setColor(Color.WHITE);
166             } else {
167                 g.setColor(DXF_Color.getColor(c));
168             }
169         } catch (NullPointerException e) {
170             g.setColor(DXF_Color.getColor(c));
171         }
172
173         if (DXF_Loader.checkLineType) {
174             if (_lineType != null
175                     && _lineType._name.equalsIgnoreCase("BYLAYER")) {
176                 if (_refLayer != null) {
177                     _lineStroke = _refLayer._stroke;
178                 }
179             } else {
180                 _lineStroke = _stroke;
181             }
182             if (_lineStroke == null)
183                 _lineStroke = myTable.defaultStroke;
184
185             if (_lineStroke != null) {
186                 if (((Graphics2D) g).getStroke() != _lineStroke)
187                     ((Graphics2D) g).setStroke(_lineStroke);
188             }
189         }
190
191     }
192
193     public Vector<DefaultMutableTreeNode> getCommonNode() {
194         Vector<DefaultMutableTreeNode> v = new Vector<DefaultMutableTreeNode>();
195
196         v.add(new DefaultMutableTreeNode(new myLabel(myLabel.COLOR, String
197                 .valueOf(this._color))));
198
199         if (isVisible)
200             v.add(new DefaultMutableTreeNode(new myLabel(myLabel.VISISBILITY,
201                     String.valueOf(0))));
202         else
203             v.add(new DefaultMutableTreeNode(new myLabel(myLabel.VISISBILITY,
```

```
                    String.valueOf(1))));

        if (_lineType != null) {
            v.add(new DefaultMutableTreeNode(new myLabel(myLabel.TYPE_LIGNE,
                    _lineType._name)));
        }
        return v;
    }

    public void writeCommon(FileWriter out) throws IOException {
        if (_color > 0) {
            out.write("62\n");
            out.write(_color + "\n");
        }
        if (_refLayer != null) {
            out.write("8\n");
            out.write(_refLayer._nom + "\n");
        }
        if (_lineType != null && !(this instanceof myBlock)
                && !(this instanceof myPoint)) {
            out.write("6\n");
            out.write(_lineType._name + "\n");
        }

        if (!(this instanceof myLayer) || !(this instanceof myBlock)) {
            out.write("60\n");
            if (isVisible)
                out.write("0\n");
            else
                out.write("1\n");
        }
    }

    public void setVisible(boolean bool) {
        isVisible = bool;
    }

    public String getIconName(myLabel Value) {
        String nomImage = "spacer.gif";
        if (Value._code.equalsIgnoreCase(myLabel.X)) {
            nomImage = "x.gif";
        } else if (Value._code.equalsIgnoreCase(myLabel.Y)) {
            nomImage = "y.gif";
        } else if (Value._code.equalsIgnoreCase(myLabel.XA)) {
            nomImage = "xa.gif";
        } else if (Value._code.equalsIgnoreCase(myLabel.YA)) {
            nomImage = "ya.gif";
        } else if (Value._code.equalsIgnoreCase(myLabel.XB)) {
            nomImage = "xb.gif";
        } else if (Value._code.equalsIgnoreCase(myLabel.YB)) {
            nomImage = "yb.gif";
        } else if (Value._code.equalsIgnoreCase(myLabel.XC)) {
            nomImage = "x.gif";
        } else if (Value._code.equalsIgnoreCase(myLabel.YC)) {
            nomImage = "y.gif";
        } else if (Value._code.equalsIgnoreCase(myLabel.XD)) {
            nomImage = "x.gif";
        } else if (Value._code.equalsIgnoreCase(myLabel.YD)) {
            nomImage = "y.gif";
        } else if (Value._code.equalsIgnoreCase(myLabel.COLOR)) {
            nomImage = "color.gif";
        } else if (Value._code.equalsIgnoreCase(myLabel.RADIUS)) {
```

```
266             nomImage = "radius.gif";
267         } else if (Value._code.equalsIgnoreCase(myLabel.BULGE)) {
268             nomImage = "bezier.gif";
269         } else if (Value._code.equalsIgnoreCase(myLabel.TYPE_LIGNE)) {
270             nomImage = "type_ligne.gif";
271         } else if (Value._code.equalsIgnoreCase(myLabel.ANGLE1)) {
272             nomImage = "angle.gif";
273         } else if (Value._code.equalsIgnoreCase(myLabel.ANGLE2)) {
274             nomImage = "angle.gif";
275         } else if (Value._code.equalsIgnoreCase(myLabel.ROTATION)) {
276             nomImage = "rotation.gif";
277         } else if (Value._code.equalsIgnoreCase(myLabel.ALIGN)) {
278             nomImage = "align.gif";
279         } else if (Value._code.equalsIgnoreCase(myLabel.HEIGHT)) {
280             nomImage = "height.gif";
281         } else if (Value._code.equalsIgnoreCase(myLabel.THICKNESS)) {
282             nomImage = "thickness.png";
283         } else if (Value._code.equalsIgnoreCase(myLabel.VALUE)) {
284             nomImage = "value.gif";
285         } else if (Value._code.equalsIgnoreCase(myLabel.FLAG)) {
286             nomImage = "flag.gif";
287         } else if (Value._code.equalsIgnoreCase(myLabel.STYLE)) {
288             nomImage = "style.gif";
289         } else if (Value._code.equalsIgnoreCase(myLabel.VISISBILITY)) {
290             nomImage = "visible.gif";
291         } else if (Value._code.equalsIgnoreCase(myLabel.ZOOM_FACTOR)) {
292             nomImage = "zoom.gif";
293         } else if (Value._code.equalsIgnoreCase(myLabel.BLOCK)) {
294             nomImage = "block.png";
295         }
296         return ("images/" + nomImage);
297     }
298 
299     public void setSelected(boolean s) {
300         this.selected = s;
301     }
302 
303     public void setChanging(boolean b) {
304         this.changing = b;
305     }
306 
307     public void obtenerDatos() {
308 
309     };
310 }
311 
```

```
 1 /*-------------------------------------------------------------------------------------
 2 Copyright 2007, Stephan Soulard and Edouard Vanhauwaert.
 3 Copyright 2014, Celeste Gabriela Guagliano.
 4
 5 This file was originaly part of DXF project and then modified by
 6 Celeste Gabriela Guagliano for DXF2Machine project.
 7
 8 DXF2Machine is free software: you can redistribute it and/or modify it under the terms of
 9 the GNU General Public License as published by the Free Software Foundation, either
10 version 2 of the License.
11
12 DXF2Machine is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
13 without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
14 See the GNU General Public License for more details.
15
16 You should have received a copy of the GNU General Public License along with DXF2Machine.
17 If not, see <http://www.gnu.org/licenses/>.
18   -------------------------------------------------------------------------------------*/
19
20 /*
21  * Initials     Name
22  * ------------------------------------
23  * CeGu         Celeste Guagliano.
24  */
25
26 /*
27  * modification history (new versions first)
28  * -----------------------------------------------------------
29  * 20141002 v0.0.2 CeGu add obtenerDatos() method
30  * 20140428 v0.0.1 CeGu fork from DXF project
31
32 */
33
34 package myDXF.Entities;
35
36 import java.awt.Graphics;
60 /**
61  * This class to manage the line's data.
62  * @author: Stephan Soulard, Edouard Vanhauwaert
63  * @version: 13/01/15 by Celeste Guagliano
64  *
65  */
66 public class myLine extends myEntity {
67
68     private static final long serialVersionUID = 1L;
69     public myPoint _a = new myPoint();
70     public myPoint _b = new myPoint();
71     private Line2D.Double _l = new Line2D.Double();
72     public boolean ubicado = false;
73     public int posicion = 0;
74     static DecimalFormat formatoNumero = (DecimalFormat) DecimalFormat.getNumberInstance();
75
76     public myLine(myPoint a, myPoint b, int c, myLayer l, myLineType lineType,
77             double thickness, int visibility) {
78         super(c, l, visibility, lineType, thickness);
79         _a = a;
80         _b = b;
81         // myDXF.Entities.TablaLineas.Tabla(datos);
82         myStats.nbLine += 1;
83
84     }
85
```

```
86      @Override
87      public void obtenerDatos() {
88          double Xinicial = _a.X();
89          double Yinicial = _a.Y();
90          double Xfinal = _b.X();
91          double Yfinal = _b.Y();
92          int orientacion = 0;
93          Xinicial=(double) FormatoNumeros.formatearNumero(Xinicial);
94          Yinicial=(double) FormatoNumeros.formatearNumero(Yinicial);
95          Xfinal=(double) FormatoNumeros.formatearNumero(Xfinal);
96          Yfinal=(double) FormatoNumeros.formatearNumero(Yfinal);
97          datos datos = new DatosLinea(Xinicial,Yinicial,Xfinal,Yfinal, _color, ubicado,
    posicion, orientacion);
98          Tabla.agregarDatoATabla(datos);
99      }
100
101     public myLine() {
102         super(-1, null, 0, null, myTable.defaultThickness);
103         myStats.nbLine += 1;
104     }
105
106     public myLine(myLine original) {
107         super(original._color, original._refLayer, 0, original._lineType,
108                 original._thickness);
109         _a = new myPoint(original._a);
110         _b = new myPoint(original._b);
111
112         myStats.nbLine += 1;
113     }
114
115     @Override
116     public void draw(Graphics g) {
117
118         _l.setLine(myCoord.dxfToJava_X(_a.X()), myCoord.dxfToJava_Y(_a.Y()),
119                 myCoord.dxfToJava_X(_b.X()), myCoord.dxfToJava_Y(_b.Y()));
120
121         super.draw(g);
122         ((Graphics2D) g).draw(_l);
123     }
124
125     @Override
126     public Line2D.Double getSelectedEntity() {
127         return _l;
128     }
129
130     public static myLine read(myBufferedReader br, myUnivers univers)
131             throws IOException {
132         String ligne = "", ligne_temp = "";
133         myLayer l = null;
134         double x1 = 0, y1 = 0, x2 = 0, y2 = 0, thickness = 0;
135         myLineType lineType = null;
136         int visibility = 0, c = -1;
137         myLog.writeLog("> new myLine");
138
139         while ((ligne = br.readLine()) != null
140                 && !ligne.trim().equalsIgnoreCase("0")) {
141             ligne_temp = ligne;
142             ligne = br.readLine();
143
144             if (ligne_temp.equalsIgnoreCase("10")) {
145                 x1 = Double.parseDouble(ligne);
146             } else if (ligne_temp.equalsIgnoreCase("20")) {
```

```
147                 y1 = Double.parseDouble(ligne);
148             } else if (ligne_temp.equalsIgnoreCase("11")) {
149                 x2 = Double.parseDouble(ligne);
150             } else if (ligne_temp.equalsIgnoreCase("21")) {
151                 y2 = Double.parseDouble(ligne);
152             } else if (ligne_temp.equalsIgnoreCase("8")) {
153                 l = univers.findLayer(ligne);
154             } else if (ligne_temp.equalsIgnoreCase("62")) {
155                 c = Integer.parseInt(ligne);
156             } else if (ligne_temp.equalsIgnoreCase("6")) {
157                 lineType = univers.findLType(ligne);
158             } else if (ligne_temp.equalsIgnoreCase("39")) {
159                 thickness = Double.parseDouble(ligne);
160                 ;
161             } else if (ligne_temp.equalsIgnoreCase("60")) {
162                 visibility = Integer.parseInt(ligne);
163             } else {
164                 myLog.writeLog("Unknown :" + ligne_temp + "(" + ligne + ")");
165             }
166         }
167         return new myLine(new myPoint(x1, y1, c, l, visibility, 1),
168                 new myPoint(x2, y2, c, l, visibility, 1), c, l, lineType,
169                 thickness, visibility);
170     }
171 
172     @Override
173     public void write(FileWriter out) throws IOException {
174         out.write("LINE\n");
175         out.write("10\n");
176         out.write(_a.X() + "\n");
177         out.write("20\n");
178         out.write(_a.Y() + "\n");
179         out.write("11\n");
180         out.write(_b.X() + "\n");
181         out.write("21\n");
182         out.write(_b.Y() + "\n");
183         out.write("6\n");
184         out.write(_lineType + "\n");
185         out.write("39\n");
186         out.write(_thickness + "\n");
187         super.writeCommon(out);
188         out.write("0\n");
189     }
190 
191     @Override
192     public String toString() {
193         return DXF_Loader.res.getString("myLine.0");
194     }
195 
196     @Override
197     public DefaultMutableTreeNode getNode() {
198 
199         DefaultMutableTreeNode root = new DefaultMutableTreeNode(this);
200 
201         root.add(new DefaultMutableTreeNode(new myLabel(myLabel.XA, String
202                 .valueOf(this._a.X()))));
203         root.add(new DefaultMutableTreeNode(new myLabel(myLabel.YA, String
204                 .valueOf(this._a.Y()))));
205         root.add(new DefaultMutableTreeNode(new myLabel(myLabel.XB, String
206                 .valueOf(this._b.X()))));
207         root.add(new DefaultMutableTreeNode(new myLabel(myLabel.YB, String
208                 .valueOf(this._b.Y()))));
```

```
209         root.add(new DefaultMutableTreeNode(new myLabel(myLabel.THICKNESS,
210                 String.valueOf(this._thickness))));
211
212         Vector<DefaultMutableTreeNode> v = super.getCommonNode();
213
214         for (int i = 0; i < v.size(); i++)
215             root.add(v.get(i));
216         return root;
217     }
218
219     @Override
220     public myLabel getNewLabel(String code, Object newValue)
221             throws NumberFormatException {
222         myLabel l = null;
223
224         if (code.equals(myLabel.XA)) {
225             _a.setX(Double.parseDouble(newValue.toString()));
226             l = new myLabel(myLabel.XA, newValue.toString());
227         } else if (code.equals(myLabel.YA)) {
228             _a.setY(Double.parseDouble(newValue.toString()));
229             l = new myLabel(myLabel.YA, newValue.toString());
230         } else if (code.equals(myLabel.XB)) {
231             _b.setX(Double.parseDouble(newValue.toString()));
232             l = new myLabel(myLabel.XB, newValue.toString());
233         } else if (code.equals(myLabel.YB)) {
234             _b.setY(Double.parseDouble(newValue.toString()));
235             l = new myLabel(myLabel.YB, newValue.toString());
236         } else if (code.equals(myLabel.THICKNESS)) {
237             _thickness = Double.parseDouble(newValue.toString());
238             l = new myLabel(myLabel.THICKNESS, newValue.toString());
239         } else {
240             l = super.getCommonLabel(code, newValue);
241         }
242         return l;
243     }
244
245     @Override
246     public void translate(double x, double y) {
247         this._a._point.x -= myCoord.getTransalation(x);
248         this._a._point.y += myCoord.getTransalation(y);
249         this._b._point.x -= myCoord.getTransalation(x);
250         this._b._point.y += myCoord.getTransalation(y);
251     }
252
253     @Override
254     public double getMinX(double min) {
255         if (_a.X() < min)
256             min = _a.X();
257         if (_b.X() < min)
258             min = _b.X();
259         return min;
260     }
261
262     @Override
263     public double getMaxX(double max) {
264         if (_a.X() > max)
265             max = _a.X();
266         if (_b.X() > max)
267             max = _b.X();
268         return max;
269     }
270
```

```java
    @Override
    public double getMinY(double min) {
        if (_a.Y() < min)
            min = _a.Y();
        if (_b.Y() < min)
            min = _b.Y();
        return min;
    }

    @Override
    public double getMaxY(double max) {
        if (_a.Y() > max)
            max = _a.Y();
        if (_b.Y() > max)
            max = _b.Y();

        return max;
    }

    @Override
    protected void finalize() throws Throwable {
        myStats.nbLine -= 1;
        super.finalize();
    }

}

```

```java
/*-------------------------------------------------------------------------------------
Copyright 2007, Stephan Soulard and Edouard Vanhauwaert.
Copyright 2014, Celeste Gabriela Guagliano.

This file was originaly part of DXF project and then modified by
Celeste Gabriela Guagliano for DXF2Machine project.

DXF2Machine is free software: you can redistribute it and/or modify it under the terms of
the GNU General Public License as published by the Free Software Foundation, either
version 2 of the License.

DXF2Machine is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with DXF2Machine.
If not, see <http://www.gnu.org/licenses/>.
  -------------------------------------------------------------------------------------*/

/*
 * Initials     Name
 * ------------------------------------
 * CeGu         Celeste Guagliano.
 */

/*
 * modification history (new versions first)
 * -----------------------------------------------------------
 * 20150113 v0.0.2 CeGu modified ColorMap
 * 20140428 v0.0.1 CeGu fork from DXF project

*/

package myDXF.Graphics;

import java.awt.Color;

/**
 * Class to generate a color map
 * @author: Stephan Soulard, Edouard Vanhauwaert
 * @version: 13/01/15 by Celeste Guagliano
 *
 */
public final class DXF_Color {
    private static int defaultColor = 0;

    static final Color[] ColorMap =
        {
            new Color (0xff, 0xff, 0xff),     // color 255 white
            new Color (0xff, 0x00, 0x00),     // color 1    Red
            new Color (0xff, 0xff, 0x00),     // color 2    Yellow
            new Color (0x00, 0xff, 0x00),     // color 3    Green
            new Color (0x00, 0xff, 0xff),     // color 4    Cyan
            new Color (0x00, 0x00, 0xff),     // color 5    Blue
            new Color (0xff, 0x00, 0xff),     // color 6    Magenta
            new Color (0xff, 0xff, 0xff),     // color 7    White
            new Color (0x98, 0x98, 0x98),     // color 8    Grey
            new Color (0xc0, 0xc0, 0xc0),     // color 9
        //  new Color (0xff, 0x00, 0x00),     // color 10
            new Color (0xff, 0x80, 0x80),     // color 11
            new Color (0xa6, 0x00, 0x00),     // color 12
            new Color (0xa6, 0x53, 0x53),     // color 13
```

```
 63            new Color (0x80, 0x00, 0x00),     // color 14
 64            new Color (0x80, 0x40, 0x40),     // color 15
 65            new Color (0x4c, 0x00, 0x00),     // color 16
 66            new Color (0x4c, 0x26, 0x26),     // color 17
 67            new Color (0x26, 0x00, 0x00),     // color 18
 68            new Color (0x26, 0x13, 0x13),     // color 19
 69            new Color (0xff, 0x40, 0x00),     // color 20
 70            new Color (0xff, 0x9f, 0x80),     // color 21
 71            new Color (0xa6, 0x29, 0x00),     // color 22
 72            new Color (0xa6, 0x68, 0x53),     // color 23
 73            new Color (0x80, 0x20, 0x00),     // color 24
 74            new Color (0x80, 0x50, 0x40),     // color 25
 75            new Color (0x4c, 0x13, 0x00),     // color 26
 76            new Color (0x4c, 0x30, 0x26),     // color 27
 77            new Color (0x26, 0x0a, 0x00),     // color 28
 78            new Color (0x26, 0x18, 0x13),     // color 29
 79            new Color (0xff, 0x80, 0x00),     // color 30
 80            new Color (0xff, 0xbf, 0x80),     // color 31
 81            new Color (0xa6, 0x53, 0x00),     // color 32
 82            new Color (0xa6, 0x7c, 0x53),     // color 33
 83            new Color (0x80, 0x40, 0x00),     // color 34
 84            new Color (0x80, 0x60, 0x40),     // color 35
 85            new Color (0x4c, 0x26, 0x00),     // color 36
 86            new Color (0x4c, 0x39, 0x26),     // color 37
 87            new Color (0x26, 0x13, 0x00),     // color 38
 88            new Color (0x26, 0x1d, 0x13),     // color 39
 89            new Color (0xff, 0xbf, 0x00),     // color 40
 90            new Color (0xff, 0xdf, 0x80),     // color 41
 91            new Color (0xa6, 0x7c, 0x00),     // color 42
 92            new Color (0xa6, 0x91, 0x53),     // color 43
 93            new Color (0x80, 0x60, 0x00),     // color 44
 94            new Color (0x80, 0x70, 0x40),     // color 45
 95            new Color (0x4c, 0x39, 0x00),     // color 46
 96            new Color (0x4c, 0x43, 0x26),     // color 47
 97            new Color (0x26, 0x1d, 0x00),     // color 48
 98            new Color (0x26, 0x21, 0x13),     // color 49
 99        //  new Color (0xff, 0xff, 0x00),     // color 50
100            new Color (0xff, 0xff, 0x80),     // color 51
101            new Color (0xa6, 0xa6, 0x00),     // color 52
102            new Color (0xa6, 0xa6, 0x53),     // color 53
103            new Color (0x80, 0x80, 0x00),     // color 54
104            new Color (0x80, 0x80, 0x40),     // color 55
105            new Color (0x4c, 0x4c, 0x00),     // color 56
106            new Color (0x4c, 0x4c, 0x26),     // color 57
107            new Color (0x26, 0x26, 0x00),     // color 58
108            new Color (0x26, 0x26, 0x13),     // color 59
109            new Color (0xbf, 0xff, 0x00),     // color 60
110            new Color (0xdf, 0xff, 0x80),     // color 61
111            new Color (0x7c, 0xa6, 0x00),     // color 62
112            new Color (0x91, 0xa6, 0x53),     // color 63
113            new Color (0x60, 0x80, 0x00),     // color 64
114            new Color (0x70, 0x80, 0x40),     // color 65
115            new Color (0x39, 0x4c, 0x00),     // color 66
116            new Color (0x43, 0x4c, 0x26),     // color 67
117            new Color (0x1d, 0x26, 0x00),     // color 68
118            new Color (0x21, 0x26, 0x13),     // color 69
119            new Color (0x80, 0xff, 0x00),     // color 70
120            new Color (0xbf, 0xff, 0x80),     // color 71
121            new Color (0x53, 0xa6, 0x00),     // color 72
122            new Color (0x7c, 0xa6, 0x53),     // color 73
123            new Color (0x40, 0x80, 0x00),     // color 74
124            new Color (0x60, 0x80, 0x40),     // color 75
```

```
125             new Color (0x26, 0x4c, 0x00),     // color 76
126             new Color (0x39, 0x4c, 0x26),     // color 77
127             new Color (0x13, 0x26, 0x00),     // color 78
128             new Color (0x1d, 0x26, 0x13),     // color 79
129             new Color (0x40, 0xff, 0x00),     // color 80
130             new Color (0x9f, 0xff, 0x80),     // color 81
131             new Color (0x29, 0xa6, 0x00),     // color 82
132             new Color (0x68, 0xa6, 0x53),     // color 83
133             new Color (0x20, 0x80, 0x00),     // color 84
134             new Color (0x50, 0x80, 0x40),     // color 85
135             new Color (0x13, 0x4c, 0x00),     // color 86
136             new Color (0x30, 0x4c, 0x26),     // color 87
137             new Color (0x0a, 0x26, 0x00),     // color 88
138             new Color (0x18, 0x26, 0x13),     // color 89
139         //  new Color (0x00, 0xff, 0x00),     // color 90
140             new Color (0x80, 0xff, 0x80),     // color 91
141             new Color (0x00, 0xa6, 0x00),     // color 92
142             new Color (0x53, 0xa6, 0x53),     // color 93
143             new Color (0x00, 0x80, 0x00),     // color 94
144             new Color (0x40, 0x80, 0x40),     // color 95
145             new Color (0x00, 0x4c, 0x00),     // color 96
146             new Color (0x26, 0x4c, 0x26),     // color 97
147             new Color (0x00, 0x26, 0x00),     // color 98
148             new Color (0x13, 0x26, 0x13),     // color 99
149             new Color (0x00, 0xff, 0x40),     // color 100
150             new Color (0x80, 0xff, 0x9f),     // color 101
151             new Color (0x00, 0xa6, 0x29),     // color 102
152             new Color (0x53, 0xa6, 0x68),     // color 103
153             new Color (0x00, 0x80, 0x20),     // color 104
154             new Color (0x40, 0x80, 0x50),     // color 105
155             new Color (0x00, 0x4c, 0x13),     // color 106
156             new Color (0x26, 0x4c, 0x30),     // color 107
157             new Color (0x00, 0x26, 0x0a),     // color 108
158             new Color (0x13, 0x26, 0x18),     // color 109
159             new Color (0x00, 0xff, 0x80),     // color 110
160             new Color (0x80, 0xff, 0xbf),     // color 111
161             new Color (0x00, 0xa6, 0x53),     // color 112
162             new Color (0x53, 0xa6, 0x7c),     // color 113
163             new Color (0x00, 0x80, 0x40),     // color 114
164             new Color (0x40, 0x80, 0x60),     // color 115
165             new Color (0x00, 0x4c, 0x26),     // color 116
166             new Color (0x26, 0x4c, 0x39),     // color 117
167             new Color (0x00, 0x26, 0x13),     // color 118
168             new Color (0x13, 0x26, 0x1d),     // color 119
169             new Color (0x00, 0xff, 0xbf),     // color 120
170             new Color (0x80, 0xff, 0xdf),     // color 121
171             new Color (0x00, 0xa6, 0x7c),     // color 122
172             new Color (0x53, 0xa6, 0x91),     // color 123
173             new Color (0x00, 0x80, 0x60),     // color 124
174             new Color (0x40, 0x80, 0x70),     // color 125
175             new Color (0x00, 0x4c, 0x39),     // color 126
176             new Color (0x26, 0x4c, 0x43),     // color 127
177             new Color (0x00, 0x26, 0x1d),     // color 128
178             new Color (0x13, 0x26, 0x21),     // color 129
179         //  new Color (0x00, 0xff, 0xff),     // color 130
180             new Color (0x80, 0xff, 0xff),     // color 131
181             new Color (0x00, 0xa6, 0xa6),     // color 132
182             new Color (0x53, 0xa6, 0xa6),     // color 133
183             new Color (0x00, 0x80, 0x80),     // color 134
184             new Color (0x40, 0x80, 0x80),     // color 135
185             new Color (0x00, 0x4c, 0x4c),     // color 136
186             new Color (0x26, 0x4c, 0x4c),     // color 137
```

```
187             new Color (0x00, 0x26, 0x26),     // color 138
188             new Color (0x13, 0x26, 0x26),     // color 139
189             new Color (0x00, 0xbf, 0xff),     // color 140
190             new Color (0x80, 0xdf, 0xff),     // color 141
191             new Color (0x00, 0x7c, 0xa6),     // color 142
192             new Color (0x53, 0x91, 0xa6),     // color 143
193             new Color (0x00, 0x60, 0x80),     // color 144
194             new Color (0x40, 0x70, 0x80),     // color 145
195             new Color (0x00, 0x39, 0x4c),     // color 146
196             new Color (0x26, 0x43, 0x4c),     // color 147
197             new Color (0x00, 0x1d, 0x26),     // color 148
198             new Color (0x13, 0x21, 0x26),     // color 149
199             new Color (0x00, 0x80, 0xff),     // color 150
200             new Color (0x80, 0xbf, 0xff),     // color 151
201             new Color (0x00, 0x53, 0xa6),     // color 152
202             new Color (0x53, 0x7c, 0xa6),     // color 153
203             new Color (0x00, 0x40, 0x80),     // color 154
204             new Color (0x40, 0x60, 0x80),     // color 155
205             new Color (0x00, 0x26, 0x4c),     // color 156
206             new Color (0x26, 0x39, 0x4c),     // color 157
207             new Color (0x00, 0x13, 0x26),     // color 158
208             new Color (0x13, 0x1d, 0x26),     // color 159
209             new Color (0x00, 0x40, 0xff),     // color 160
210             new Color (0x80, 0x9f, 0xff),     // color 161
211             new Color (0x00, 0x29, 0xa6),     // color 162
212             new Color (0x53, 0x68, 0xa6),     // color 163
213             new Color (0x00, 0x20, 0x80),     // color 164
214             new Color (0x40, 0x50, 0x80),     // color 165
215             new Color (0x00, 0x13, 0x4c),     // color 166
216             new Color (0x26, 0x30, 0x4c),     // color 167
217             new Color (0x00, 0x0a, 0x26),     // color 168
218             new Color (0x13, 0x18, 0x26),     // color 169
219         //  new Color (0x00, 0x00, 0xff),     // color 170
220             new Color (0x80, 0x80, 0xff),     // color 171
221             new Color (0x00, 0x00, 0xa6),     // color 172
222             new Color (0x53, 0x53, 0xa6),     // color 173
223             new Color (0x00, 0x00, 0x80),     // color 174
224             new Color (0x40, 0x40, 0x80),     // color 175
225             new Color (0x00, 0x00, 0x4c),     // color 176
226             new Color (0x26, 0x26, 0x4c),     // color 177
227             new Color (0x00, 0x00, 0x26),     // color 178
228             new Color (0x13, 0x13, 0x26),     // color 179
229             new Color (0x40, 0x00, 0xff),     // color 180
230             new Color (0x9f, 0x80, 0xff),     // color 181
231             new Color (0x29, 0x00, 0xa6),     // color 182
232             new Color (0x68, 0x53, 0xa6),     // color 183
233             new Color (0x20, 0x00, 0x80),     // color 184
234             new Color (0x50, 0x40, 0x80),     // color 185
235             new Color (0x13, 0x00, 0x4c),     // color 186
236             new Color (0x30, 0x26, 0x4c),     // color 187
237             new Color (0x0a, 0x00, 0x26),     // color 188
238             new Color (0x18, 0x13, 0x26),     // color 189
239             new Color (0x80, 0x00, 0xff),     // color 190
240             new Color (0xbf, 0x80, 0xff),     // color 191
241             new Color (0x53, 0x00, 0xa6),     // color 192
242             new Color (0x7c, 0x53, 0xa6),     // color 193
243             new Color (0x40, 0x00, 0x80),     // color 194
244             new Color (0x60, 0x40, 0x80),     // color 195
245             new Color (0x26, 0x00, 0x4c),     // color 196
246             new Color (0x39, 0x26, 0x4c),     // color 197
247             new Color (0x13, 0x00, 0x26),     // color 198
248             new Color (0x1d, 0x13, 0x26),     // color 199
```

```
249             new Color (0xbf, 0x00, 0xff),     // color 200
250             new Color (0xdf, 0x80, 0xff),     // color 201
251             new Color (0x7c, 0x00, 0xa6),     // color 202
252             new Color (0x91, 0x53, 0xa6),     // color 203
253             new Color (0x60, 0x00, 0x80),     // color 204
254             new Color (0x70, 0x40, 0x80),     // color 205
255             new Color (0x39, 0x00, 0x4c),     // color 206
256             new Color (0x43, 0x26, 0x4c),     // color 207
257             new Color (0x1d, 0x00, 0x26),     // color 208
258             new Color (0x21, 0x13, 0x26),     // color 209
259             new Color (0xff, 0x00, 0xff),     // color 210
260             new Color (0xff, 0x80, 0xff),     // color 211
261             new Color (0xa6, 0x00, 0xa6),     // color 212
262             new Color (0xa6, 0x53, 0xa6),     // color 213
263             new Color (0x80, 0x00, 0x80),     // color 214
264             new Color (0x80, 0x40, 0x80),     // color 215
265             new Color (0x4c, 0x00, 0x4c),     // color 216
266             new Color (0x4c, 0x26, 0x4c),     // color 217
267             new Color (0x26, 0x00, 0x26),     // color 218
268             new Color (0x26, 0x13, 0x26),     // color 219
269             new Color (0xff, 0x00, 0xbf),     // color 220
270             new Color (0xff, 0x80, 0xdf),     // color 221
271             new Color (0xa6, 0x00, 0x7c),     // color 222
272             new Color (0xa6, 0x53, 0x91),     // color 223
273             new Color (0x80, 0x00, 0x60),     // color 224
274             new Color (0x80, 0x40, 0x70),     // color 225
275             new Color (0x4c, 0x00, 0x39),     // color 226
276             new Color (0x4c, 0x26, 0x43),     // color 227
277             new Color (0x26, 0x00, 0x1d),     // color 228
278             new Color (0x26, 0x13, 0x21),     // color 229
279             new Color (0xff, 0x00, 0x80),     // color 230
280             new Color (0xff, 0x80, 0xbf),     // color 231
281             new Color (0xa6, 0x00, 0x53),     // color 232
282             new Color (0xa6, 0x53, 0x7c),     // color 233
283             new Color (0x80, 0x00, 0x40),     // color 234
284             new Color (0x80, 0x40, 0x60),     // color 235
285             new Color (0x4c, 0x00, 0x26),     // color 236
286             new Color (0x4c, 0x26, 0x39),     // color 237
287             new Color (0x26, 0x00, 0x13),     // color 238
288             new Color (0x26, 0x13, 0x1d),     // color 239
289             new Color (0xff, 0x00, 0x40),     // color 240
290             new Color (0xff, 0x80, 0x9f),     // color 241
291             new Color (0xa6, 0x00, 0x29),     // color 242
292             new Color (0xa6, 0x53, 0x68),     // color 243
293             new Color (0x80, 0x00, 0x20),     // color 244
294             new Color (0x80, 0x40, 0x50),     // color 245
295             new Color (0x4c, 0x00, 0x13),     // color 246
296             new Color (0x4c, 0x26, 0x30),     // color 247
297             new Color (0x26, 0x00, 0x0a),     // color 248
298             new Color (0x26, 0x13, 0x18),     // color 249
299             new Color (0x54, 0x54, 0x54),     // color 250
300             new Color (0x76, 0x76, 0x76),     // color 251
301             new Color (0x98, 0x98, 0x98),     // color 252
302             new Color (0xbb, 0xbb, 0xbb),     // color 253
303             new Color (0xdd, 0xdd, 0xdd),     // color 254
304             new Color (0x00, 0x00, 0x00)      // color 0    bylayer
305     };
306
307
308     public static Color getDefaultColor() {
309         return ColorMap[defaultColor];
310     }
```

```
311
312     public static int getDefaultColorIndex() {
313         return defaultColor;
314     }
315
316     public static void setDefaultColor(int n) {
317         defaultColor = n;
318         myJColorChooser.col.setBackground(getColor(defaultColor));
319     }
320
321     public static Color getSelectingColor() {
322         return ColorMap[2];
323     }
324
325     public static Color getChangingColor() {
326         return ColorMap[3];
327     }
328
329     public final static Color getColor(int index) {
330         if ((index >= 0) && (index <= 255))
331             return ColorMap[index];
332         else
333             return null;
334
335     }
336
337     public final static int getColor(Color c) {
338         if (c == null)
339             return -1;
340
341         for (int i = ColorMap.length -1; i >= 0; i--) {
342             if (ColorMap[i].equals(c))
343                 return i;
344         }
345         return defaultColor;
346     }
347 }
348
```

```java
/*------------------------------------------------------------------------------
Copyright 2007, Stephan Soulard and Edouard Vanhauwaert.
Copyright 2014, Celeste Gabriela Guagliano.

This file was originaly part of DXF project and then modified by
Celeste Gabriela Guagliano for DXF2Machine project.

DXF2Machine is free software: you can redistribute it and/or modify it under the terms of
the GNU General Public License as published by the Free Software Foundation, either
version 2 of the License.

DXF2Machine is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with DXF2Machine.
If not, see <http://www.gnu.org/licenses/>.
 ------------------------------------------------------------------------------
 */

/*
 * Initials     Name
 * ----------------------------------
 * CeGu         Celeste Guagliano.
 */

/*
 * modification history (new versions first)
 * -----------------------------------------------------------
 * 20150113 v0.0.2 CeGu improve mouseClicked() method
 * 20150113 v0.0.2 CeGu improve mouseDragged() method
 * 20150113 v0.0.2 CeGu improve getSelectedObject() method
 * 20150113 v0.0.2 CeGu improve mousePressed() method
 * 20140428 v0.0.1 CeGu fork from DXF project

*/

package myDXF.Graphics;

import java.awt.Canvas;
//import sun.print.ProxyPrintGraphics;

/**
 * Class to manage the canvas
 * @author: Stephan Soulard, Edouard Vanhauwaert
 * @version: 13/01/15 by Celeste Guagliano
 *
 */
public class myCanvas extends Canvas implements MouseListener,
        MouseMotionListener, ComponentListener, KeyListener {

    private static final long serialVersionUID = 1L;
    public static BufferedImage bi;
    public static Graphics2D big;
    public boolean firstTime = true;
    public Rectangle area;
    public static DXF_Loader _dxf;
    public Dimension dim;
    public boolean drawingLine = false;
    public Point lastClick;
    public Point currPoint;
    public Point lastDrag;
```

```
114     public Point arcStart;
115     public Point arcStop;
116     public Point arcCenter;
117     public Point arcRadius;
118     public Point origPoint;
119     public Color currColor;
120     public boolean selecting = true;
121     public boolean moving = false;
122     public boolean zooming = false;
123     public boolean drawingCircle = false;
124     public boolean drawingPolyLineStart = false;
125     public boolean drawingPolyLineEnd = false;
126     public boolean drawingLwPolyLineStart = false;
127     public boolean drawingLwPolyLineEnd = false;
128     public boolean drawingArc = false;
129     public boolean drawingArcAngleStart = false;
130     public boolean drawingArcAngleEnd = false;
131     public boolean drawingEllipse = false;
132     public boolean drawingTrace = false;
133     public boolean drawingTxt = false;
134     public boolean drawingSolid = false;
135     public boolean xSelecting = false;
136     public Vector<myVertex> tmpVectVertex = new Vector<myVertex>();
137     public int[] tmpPolyX;
138     public int[] tmpPolyY;
139     public int[] tmpLwPolyX;
140     public int[] tmpLwPolyY;
141     public Rectangle zoomRect;
142     public static myEntity clickOn;
143     public Vector<myEntity> vectClickOn = new Vector<myEntity>();
144     private int bump = 1;
145     public Vector<myEntity> clipBoard = new Vector<myEntity>();
146     public Stroke currentStroke = myTable.defaultStroke;
147     public int quadCount = 0;
148     public Point2D.Double[] quadPt = new Point2D.Double[3];
149     private myText editText;
150     private boolean firstTxt = false;
151     private myEntity changingEnt;
152     private myCircle editCircle = null;
153     private myArc editArc = null;
154     public static double selEntityVar[] = new double[3];
155     public static int nuevoColor=0;
156 
157     public myCanvas(final DXF_Loader d) {
158         super();
159         JPopupMenu.setDefaultLightWeightPopupEnabled(false);
160         _dxf = d;
161         addMouseMotionListener(this);
162         addMouseListener(this);
163         addComponentListener(this);
164         addKeyListener(this);
165 
166         this.setBackground(myUnivers._bgColor);
167 
168         try {
169             _dxf.newDXF();
170         } catch (Exception e) {
171             e.printStackTrace();
172         }
173         this.setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
174     }
175 
```

```
176     @Override
177     public void paint(Graphics g) {
178         update(g);
179     }
180
181     @Override
182     public void update(Graphics g) {
183
184         Graphics2D g2 = null;
185         Object aliasing = RenderingHints.VALUE_ANTIALIAS_OFF;
186
187         if (myUnivers.antialiasing)
188             aliasing = RenderingHints.VALUE_ANTIALIAS_ON;
189         else
190             aliasing = RenderingHints.VALUE_ANTIALIAS_OFF;
191             g2 = ((Graphics2D) g);
192             g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, aliasing);
193
194         if (firstTime) {
195             dim = getSize();
196             myCoord.Max = dim.width;
197
198             _dxf._u._header.setLIM(myCoord.javaToDXF_X(getWidth()),
199                     myCoord.javaToDXF_X(getHeight()));
200
201             area = new Rectangle(dim);
202             bi = (BufferedImage) createImage(dim.width, dim.height);
203             big = bi.createGraphics();
204             firstTime = false;
205
206         }
207         big.clearRect(0, 0, area.width, area.height);
208
209         if (this.moving)
210             bump = 3;
211         else
212             bump = 1;
213         for (int i = 0; i < _dxf._u._myTables.size(); i++) {
214             for (int j = 0; j < _dxf._u._myTables.elementAt(i)._myLayers.size(); j++) {
215                 if (this.moving) {
216                     if (_dxf._u._myTables.elementAt(i)._myLayers.elementAt(j)._myEnt
217                             .size() < 50)
218                         bump = 1;
219                     else
220                         bump = 3;
221                 }
222                 for (int k = 0; k < _dxf._u._myTables.elementAt(i)._myLayers
223                         .elementAt(j)._myEnt.size(); k += bump) {
224                     if (_dxf._u._myTables.elementAt(i)._myLayers.elementAt(j).isVisible
225                             && _dxf._u._myTables.elementAt(i)._myLayers
226                                     .elementAt(j)._flag != 1) {
227
228                         if (_dxf._u._myTables.elementAt(i)._myLayers
229                                 .elementAt(j)._myEnt.elementAt(k).isVisible) {
230                             _dxf._u._myTables.elementAt(i)._myLayers
231                                     .elementAt(j)._myEnt.elementAt(k).draw(big);
232
233                         }
234                     }
235                 }
236             }
237         }
```

```
238
239         Color curr = _dxf._jcc.getColor();
240         if (curr.equals(myUnivers._bgColor)) {
241             if (myUnivers._bgColor.equals(Color.WHITE))
242                 big.setColor(Color.BLACK);
243             else if (myUnivers._bgColor.equals(Color.BLACK))
244                 big.setColor(Color.WHITE);
245         } else
246             big.setColor(curr);
247
248         // affichage de la ligne pdt un glisser de la souris (outil ligne)
249         if (this.drawingLine) {
250             int x1 = (int) (this.lastClick.getX());
251             int y1 = (int) (this.lastClick.getY());
252             int x2 = (int) (this.currPoint.getX());
253             int y2 = (int) (this.currPoint.getY());
254             big.drawLine(x1, y1, x2, y2);
255         } else if (drawingPolyLineStart || drawingPolyLineEnd) {
256             this.tmpPolyX = new int[tmpVectVertex.size()];
257             this.tmpPolyY = new int[tmpVectVertex.size()];
258             for (int i = 0; i < tmpPolyX.length; i++) {
259                 tmpPolyX[i] = (int) myCoord.dxfToJava_X(tmpVectVertex
260                         .elementAt(i).X());
261                 tmpPolyY[i] = (int) myCoord.dxfToJava_Y(tmpVectVertex
262                         .elementAt(i).Y());
263             }
264             big.drawPolyline(this.tmpPolyX, tmpPolyY, tmpPolyX.length);
265             big.drawLine(tmpPolyX[tmpPolyX.length - 1],
266                     tmpPolyY[tmpPolyY.length - 1], (int) currPoint.getX(),
267                     (int) currPoint.getY());
268             this.tmpPolyX = null;
269             this.tmpPolyY = null;
270         } else if (drawingLwPolyLineStart || drawingLwPolyLineEnd) {
271             this.tmpLwPolyX = new int[tmpVectVertex.size()];
272             this.tmpLwPolyY = new int[tmpVectVertex.size()];
273             for (int i = 0; i < tmpPolyX.length; i++) {
274                 tmpPolyX[i] = (int) myCoord.dxfToJava_X(tmpVectVertex
275                         .elementAt(i).X());
276                 tmpPolyY[i] = (int) myCoord.dxfToJava_Y(tmpVectVertex
277                         .elementAt(i).Y());
278             }
279             big.drawPolyline(this.tmpPolyX, tmpPolyY, tmpPolyX.length);
280             big.drawLine(tmpPolyX[tmpPolyX.length - 1],
281                     tmpPolyY[tmpPolyY.length - 1], (int) currPoint.getX(),
282                     (int) currPoint.getY());
283             this.tmpLwPolyX = null;
284             this.tmpLwPolyY = null;
285         } else if (this.drawingCircle) {
286             big.drawOval(
287                     (int) lastClick.getX()
288                             - (int) lastClick.distance(currPoint),
289                     (int) lastClick.getY()
290                             - (int) lastClick.distance(currPoint),
291                     (int) lastClick.distance(currPoint) * 2,
292                     (int) lastClick.distance(currPoint) * 2);
293         } else if (this.drawingArc) {
294             big.drawOval(
295                     (int) lastClick.getX()
296                             - (int) lastClick.distance(currPoint),
297                     (int) lastClick.getY()
298                             - (int) lastClick.distance(currPoint),
299                     (int) lastClick.distance(currPoint) * 2,
```

```
300                     (int) lastClick.distance(currPoint) * 2);
301             } else if (this.drawingArcAngleStart) {
302                 big.drawOval(
303                         (int) arcCenter.getX()
304                                 - (int) arcCenter.distance(arcRadius),
305                         (int) arcCenter.getY()
306                                 - (int) arcCenter.distance(arcRadius),
307                         (int) arcCenter.distance(arcRadius) * 2,
308                         (int) arcCenter.distance(arcRadius) * 2);
309                 big.draw(new Line2D.Double(arcCenter.getX(), arcCenter.getY(),
310                         currPoint.getX(), currPoint.y));
311             } else if (this.drawingArcAngleEnd) {
312                 big.drawOval(
313                         (int) arcCenter.getX()
314                                 - (int) arcCenter.distance(arcRadius),
315                         (int) arcCenter.getY()
316                                 - (int) arcCenter.distance(arcRadius),
317                         (int) arcCenter.distance(arcRadius) * 2,
318                         (int) arcCenter.distance(arcRadius) * 2);
319                 big.draw(new Line2D.Double(arcCenter.getX(), arcCenter.getY(),
320                         currPoint.getX(), currPoint.y));
321             } else if (this.zooming) {
322                 int x1 = 0, y1 = 0, x2 = 0, y2 = 0, min = 0, caseDraw = 0;
323                 big.setColor(Color.LIGHT_GRAY);
324                 big.setStroke(myTable.zoomStroke);
325 
326                 if ((lastClick.getX() < currPoint.getX())
327                         && (lastClick.getY() < currPoint.getY()))
328                     caseDraw = myCoord.SE;
329                 else if ((lastClick.getX() < currPoint.getX())
330                         && (lastClick.getY() > currPoint.getY()))
331                     caseDraw = myCoord.NE;
332                 else if ((lastClick.getX() > currPoint.getX())
333                         && (lastClick.getY() > currPoint.getY()))
334                     caseDraw = myCoord.NO;
335                 else if ((lastClick.getX() > currPoint.getX())
336                         && (lastClick.getY() < currPoint.getY()))
337                     caseDraw = myCoord.SO;
338 
339                 switch (caseDraw) {
340                 case myCoord.NE:
341                     x1 = (int) this.lastClick.getX();
342                     y1 = (int) this.currPoint.getY();
343                     x2 = (int) (this.currPoint.getX() - this.lastClick.getX());
344                     y2 = (int) (this.lastClick.getY() - this.currPoint.getY());
345                     break;
346 
347                 case myCoord.NO:
348                     x1 = (int) this.currPoint.getX();
349                     y1 = (int) this.currPoint.getY();
350                     x2 = (int) (this.lastClick.getX() - this.currPoint.getX());
351                     y2 = (int) (this.lastClick.getY() - this.currPoint.getY());
352                     break;
353 
354                 case myCoord.SE:
355                     x1 = (int) this.lastClick.getX();
356                     y1 = (int) this.lastClick.getY();
357                     x2 = (int) (this.currPoint.getX() - this.lastClick.getX());
358                     y2 = (int) (this.currPoint.getY() - this.lastClick.getY());
359                     break;
360 
361                 case myCoord.SO:
```

```
362                 x1 = (int) this.currPoint.getX();
363                 y1 = (int) this.lastClick.getY();
364                 x2 = (int) (this.lastClick.getX() - this.currPoint.getX());
365                 y2 = (int) (this.currPoint.getY() - this.lastClick.getY());
366                 break;
367 
368             default:
369                 break;
370             }
371             if (x2 <= y2)
372                 min = x2;
373             else
374                 min = y2;
375             zoomRect.setLocation(x1, y1);
376             zoomRect.setSize(min, min);
377             big.draw(zoomRect);
378             big.setStroke(currentStroke);
379         } else if (this.xSelecting) {
380             int x1 = 0, y1 = 0, x2 = 0, y2 = 0, min = 0, caseDraw = myCoord.SE;
381             big.setColor(Color.LIGHT_GRAY);
382 
383             if ((lastClick.getX() < currPoint.getX())
384                     && (lastClick.getY() < currPoint.getY()))
385                 caseDraw = myCoord.SE;
386             else if ((lastClick.getX() < currPoint.getX())
387                     && (lastClick.getY() > currPoint.getY()))
388                 caseDraw = myCoord.NE;
389             else if ((lastClick.getX() > currPoint.getX())
390                     && (lastClick.getY() > currPoint.getY()))
391                 caseDraw = myCoord.NO;
392             else if ((lastClick.getX() > currPoint.getX())
393                     && (lastClick.getY() < currPoint.getY()))
394                 caseDraw = myCoord.SO;
395 
396             switch (caseDraw) {
397             case myCoord.NE:
398                 x1 = (int) this.lastClick.getX();
399                 y1 = (int) this.currPoint.getY();
400                 x2 = (int) (this.currPoint.getX() - this.lastClick.getX());
401                 y2 = (int) (this.lastClick.getY() - this.currPoint.getY());
402                 break;
403 
404             case myCoord.NO:
405                 x1 = (int) this.currPoint.getX();
406                 y1 = (int) this.currPoint.getY();
407                 x2 = (int) (this.lastClick.getX() - this.currPoint.getX());
408                 y2 = (int) (this.lastClick.getY() - this.currPoint.getY());
409                 break;
410 
411             case myCoord.SE:
412                 x1 = (int) this.lastClick.getX();
413                 y1 = (int) this.lastClick.getY();
414                 x2 = (int) (this.currPoint.getX() - this.lastClick.getX());
415                 y2 = (int) (this.currPoint.getY() - this.lastClick.getY());
416                 break;
417 
418             case myCoord.SO:
419                 x1 = (int) this.currPoint.getX();
420                 y1 = (int) this.lastClick.getY();
421                 x2 = (int) (this.lastClick.getX() - this.currPoint.getX());
422                 y2 = (int) (this.currPoint.getY() - this.lastClick.getY());
423                 break;
```

```
424             }
425             if (x2 <= y2)
426                 min = x2;
427             else
428                 min = y2;
429             zoomRect.setLocation(x1, y1);
430             zoomRect.setSize(min, min);
431             big.draw(zoomRect);
432         } else if (this.drawingEllipse) {
433             big.drawOval((int) lastClick.getX(), (int) lastClick.getY(),
434                     (int) (currPoint.getX() - lastClick.getX()),
435                     (int) (currPoint.getY() - lastClick.getY()));
436         } else if (drawingTrace) {
437             for (int i = 0; i < quadCount - 1; i++) {
438                 big.drawLine((int) myCoord.dxfToJava_X(quadPt[i].x),
439                         (int) myCoord.dxfToJava_Y(quadPt[i].y),
440                         (int) myCoord.dxfToJava_X(quadPt[i + 1].x),
441                         (int) myCoord.dxfToJava_Y(quadPt[i + 1].y));
442             }
443             big.drawLine((int) (this.lastClick.getX()),
444                     (int) (this.lastClick.getY()),
445                     (int) (this.currPoint.getX()),
446                     (int) (this.currPoint.getY()));
447         }
448         repaint();
449 
450         // GLOBAL MAP
451         {
452             big.setColor(Color.YELLOW);
453             // big.setStroke(mapStroke);
454             double x1 = area.width - area.width * 0.1;
455             double y1 = area.height - area.height * 0.1;
456             double w1 = area.width * 0.1;
457             double h1 = area.height * 0.1;
458 
459             big.clearRect((int) x1, (int) y1, (int) w1, (int) h1);
460             big.drawRect((int) x1, (int) y1, (int) w1, (int) h1);
461 
462             big.setColor(Color.RED);
463 
464             double x2 = x1 + 0.1 * _dxf._u._header._LIMMIN.X();
465             ;
466             double y2 = area.height
467                     + (_dxf._u._header._LIMMIN.Y() - _dxf._u._header._LIMMAX
468                             .Y()) * 0.1;
469             double w2 = w1
470                     - (_dxf._u._header._EXTMAX.X() - _dxf._u._header._LIMMAX
471                             .X()) * 0.1;
472             double h2 = (_dxf._u._header._LIMMIN.X() - _dxf._u._header._LIMMAX
473                     .X()) * 0.1;
474 
475             big.fillRect((int) x2, (int) y2, (int) w2, (int) h2);
476         }
477         // FIN MAP
478         big.setBackground(myUnivers._bgColor);
479         g2.drawImage(bi, 0, 0, this); // double buffering
480     }
481 
482     @Override
483     public void mousePressed(MouseEvent e) {
484         this.lastClick = e.getPoint();
485         this.currPoint = e.getPoint();
```

```
486         /*
487          * if (_dxf._typeOutil.getSelectedIndex()==myToolBar.toolLine) {
488          * this.drawingLine=true; } else if
489          * (_dxf._typeOutil.getSelectedIndex()==myToolBar.toolCircle) {
490          * this.drawingCircle=true; } else if
491          * (_dxf._typeOutil.getSelectedIndex()==myToolBar.toolDeplace) {
492          */this.moving = true;
493         if (_dxf._jcc.getSelectedIndex() == myJColorChooser.toolNone) {
494             this.selecting = false;
495         }
496 
497         if (_dxf._jcc.getSelectedIndex() == myJColorChooser.toolSel) {
498             this.selecting = true;
499             if ((e.getModifiersEx() == 1024) || (e.getModifiersEx() == 1088)) { // rien
500                                                                                 // ou
501                                                                                 // shift
502                 for (int i = 0; i < vectClickOn.size(); i++)
503                     vectClickOn.elementAt(i).setSelected(false);
504                 vectClickOn = null;
505                 vectClickOn = new Vector<myEntity>();
506             }
507             getSelectedObject(e.getX(), e.getY());
508             if (clickOn != null) {
509                 boolean sel = true;
510                 if (e.getModifiersEx() == 1088) { // shift
511                     if (clickOn instanceof myLine) {
512                         Point2D.Double pA = new Point2D.Double();
513                         pA.x = myCoord.dxfToJava_X(((myLine) clickOn)._a.X());
514                         pA.y = myCoord.dxfToJava_Y(((myLine) clickOn)._a.Y());
515                         Point2D.Double pB = new Point2D.Double();
516                         pB.x = myCoord.dxfToJava_X(((myLine) clickOn)._b.X());
517                         pB.y = myCoord.dxfToJava_Y(((myLine) clickOn)._b.Y());
518                         double A = e.getPoint().distance(pA);
519                         double B = e.getPoint().distance(pB);
520                         double limits = pA.distance(pB);
521                         double maxLim = 10;
522                         try {
523                             limits /= 3;
524                         } catch (Exception ex) { // Dividebyzero
525                             limits = 1;
526                         }
527                         if (limits > maxLim)
528                             limits = maxLim;
529                         if ((A < B) && (A < limits)) {
530                             clickOn.setChanging(true);
531                             this.changingEnt = clickOn;
532                             clickOn = ((myLine) clickOn)._a;
533                             sel = false;
534                         } else if ((B <= A) && (B < limits)) {
535                             clickOn.setChanging(true);
536                             this.changingEnt = clickOn;
537                             clickOn = ((myLine) clickOn)._b;
538                             sel = false;
539                         }
540                     } else if (clickOn instanceof myCircle) {
541                         this.editCircle = ((myCircle) clickOn);
542                         this.changingEnt = clickOn;
543                         this.changingEnt.setChanging(true);
544                         this.editCircle.setChanging(true);
545                         sel = false;
546                     } else if (clickOn instanceof myPolyline) {
547                         int len = ((myPolyline) clickOn)._myVertex.size();
```

```
548                     Point2D.Double pX[] = new Point2D.Double[len];
549                     double min = Double.MAX_VALUE, dist = 0;
550                     int find = -1;
551                     for (int i = 0; i < len; i++) {
552                         pX[i] = new Point2D.Double();
553                         pX[i].x = myCoord
554                                 .dxfToJava_X(((myPolyline) clickOn)._myVertex
555                                         .elementAt(i).X());
556                         pX[i].y = myCoord
557                                 .dxfToJava_Y(((myPolyline) clickOn)._myVertex
558                                         .elementAt(i).Y());
559                         dist = pX[i].distance(e.getPoint());
560                         if (dist < min) {
561                             min = dist;
562                             find = i;
563                         }
564                     }
565                     double maxLim = 10;
566                     if (find != -1) {
567                         double d1 = Double.MAX_VALUE, d2 = Double.MAX_VALUE;
568                         int xA = 0, xB = 0;
569                         if (find == 0) {
570                             xA = 1;
571                             d1 = pX[0].distance(pX[xA]);
572                             xB = len - 1;
573                             d2 = pX[0].distance(pX[xB]);
574                         } else if (find == (len - 1)) {
575                             xA = 0;
576                             d1 = pX[len - 1].distance(pX[0]);
577                             xB = find - 1;
578                             d2 = pX[len - 1].distance(pX[xB]);
579                         } else {
580                             xA = find + 1;
581                             d1 = pX[find].distance(pX[find + 1]);
582                             xB = find - 1;
583                             d2 = pX[find].distance(pX[xB]);
584                         }
585                         double limits = 1;
586                         if ((d1 != 0) && (d2 != 0)) {
587                             if (d1 < d2) {
588                                 limits = d1 / 3;
589                                 dist = pX[xA].distance(e.getPoint());
590                             } else {
591                                 limits = d2 / 3;
592                                 dist = pX[xB].distance(e.getPoint());
593                             }
594                             if (limits > maxLim)
595                                 limits = maxLim;
596                         }
597                         if ((e.getPoint().distance(pX[find]) < limits)) {
598                             this.changingEnt = (clickOn);
599                             this.changingEnt.setChanging(true);
600                             clickOn = ((myPolyline) clickOn)._myVertex
601                                     .elementAt(find);
602                             sel = false;
603                         }
604                     }
605
606                 } else if (clickOn instanceof myArc) {
607                     this.editArc = ((myArc) clickOn);
608                     this.changingEnt = clickOn;
609                     this.changingEnt.setChanging(true);
```

```
610                     this.editArc.setChanging(true);
611                     sel = false;
612                 } else if (clickOn instanceof myText) {
613 
614                     this.changingEnt = (clickOn);
615                     this.changingEnt.setChanging(true);
616                     sel = false;
617                     editText = ((myText) clickOn);
618                     drawingTxt = true;
619                     firstTxt = false;
620 
621                 }/*
622                  * else if (clickOn instanceof mySolid) {
623                  * ((mySolid)clickOn).translate(x,y); } else if (clickOn
624                  * instanceof myBlockReference) {
625                  * ((myBlockReference)clickOn).translate(x,y); }
626                  */
627             }
628             if (!sel) {
629                 vectClickOn = null;
630                 vectClickOn = new Vector<myEntity>();
631                 vectClickOn.add(clickOn);
632             } else {
633                 if (!clickOn.selected)
634                     vectClickOn.add(clickOn);
635                 else
636                     vectClickOn.remove(clickOn);
637                 clickOn.setSelected(!clickOn.selected);
638             }
639         }
640         _dxf.sel.setText(_dxf.defSelTxtA + vectClickOn.size() + _dxf.txtB);
641         _dxf.clipB.setText(_dxf.defClipTxtA + clipBoard.size() + _dxf.txtB);
642 
643     } /*
644      * else if (_dxf._typeOutil.getSelectedIndex()==myToolBar.toolSelX) {
645      * xSelecting=true; zoomRect=new Rectangle(); } else if
646      * (_dxf._typeOutil.getSelectedIndex()==myToolBar.toolZoom) {
647      * this.zooming=true; currentStroke = big.getStroke(); zoomRect=new
648      * Rectangle(); } else if
649      * (_dxf._typeOutil.getSelectedIndex()==myToolBar.toolArc) { if
650      * (!this.drawingArc && !drawingArcAngleStart && !drawingArcAngleEnd) {
651      * this.drawingArc=true; } else if (!this.drawingArc &&
652      * drawingArcAngleStart && !drawingArcAngleEnd) { arcStart=e.getPoint();
653      * drawingArcAngleStart=false; drawingArcAngleEnd=true; } } else if
654      * (_dxf._typeOutil.getSelectedIndex()==myToolBar.toolEllipse) {
655      * this.drawingEllipse=true; }
656      */
657     _dxf.tree.updateSelection();
658     repaint();
659 }
660 
661 public void getSelectedObject(double x, double y) {
662     double min = Double.MAX_VALUE, calcMin = 0;
663     myEntity testObj = null;
664     myEntity closestObj = null;
665     clickOn = null;
666     Point2D.Double clickPoint = new Point2D.Double();
667     clickPoint.setLocation(x, y);
668     for (int i = 0; i < _dxf._u._myTables.size(); i++) {
669         for (int j = 0; j < _dxf._u._myTables.elementAt(i)._myLayers.size(); j++) {
670             for (int k = 0; k < _dxf._u._myTables.elementAt(i)._myLayers
671                     .elementAt(j)._myEnt.size(); k++) {
```

```
672                     testObj = _dxf._u._myTables.elementAt(i)._myLayers
673                             .elementAt(j)._myEnt.elementAt(k);
674                     if (testObj instanceof myPoint) {
675                         Rectangle2D.Double p = ((myPoint) testObj)
676                                 .getSelectedEntity();
677                         if ((p != null) && (p.intersects(x, y, 5, 5))) {
678                             clickOn = testObj;
679                             return;
680                         }
681                         calcMin = ((myPoint) testObj)._point
682                                 .distance(clickPoint);
683                         if (calcMin < min) {
684                             min = calcMin;
685                             closestObj = testObj;
686                         }
687                     } else if (testObj instanceof myText) {
688                         Rectangle2D.Double r = ((myText) testObj)
689                                 .getSelectedEntity();
690                         if ((r != null) && (r.intersects(x, y, 5, 10))) {
691                             clickOn = testObj;
692                             return;
693                         }
694                         calcMin = ((myText) testObj)._point._point
695                                 .distance(clickPoint);
696                         if (calcMin < min) {
697                             min = calcMin;
698                             closestObj = testObj;
699                         }
700                     } else if (testObj instanceof mySolid) {
701                         GeneralPath p = ((mySolid) testObj).getSelectedEntity();
702                         if ((p != null) && (p.intersects(x, y, 5, 5))) {
703                             clickOn = testObj;
704                             return;
705                         }
706                         calcMin = ((mySolid) testObj)._p1._point
707                                 .distance(clickPoint);
708                         if (calcMin < min) {
709                             min = calcMin;
710                             closestObj = testObj;
711                         }
712                         calcMin = ((mySolid) testObj)._p2._point
713                                 .distance(clickPoint);
714                         if (calcMin < min) {
715                             min = calcMin;
716                             closestObj = testObj;
717                         }
718                         calcMin = ((mySolid) testObj)._p3._point
719                                 .distance(clickPoint);
720                         if (calcMin < min) {
721                             min = calcMin;
722                             closestObj = testObj;
723                         }
724                         if (((mySolid) testObj)._p4 != null) {
725                             calcMin = ((mySolid) testObj)._p4._point
726                                     .distance(clickPoint);
727                             if (calcMin < min) {
728                                 min = calcMin;
729                                 closestObj = testObj;
730                             }
731                         }
732                     } else if (testObj instanceof myLine) {
733                         Line2D.Double l = ((myLine) testObj)
```

```
734                                 .getSelectedEntity();
735                         if ((l != null) && (l.intersects(x, y, 5, 5))) {
736                             clickOn = testObj;
737 
    nuevoColor=DXF_Color.getColor(myJColorChooser.col.getBackground());
738                                 _dxf._u._myTables.elementAt(i)._myLayers
739                                 .elementAt(j)._myEnt.elementAt(k)._color = nuevoColor;
740                             repaint();
741                             return;
742                         }
743                         calcMin = l.getP1().distance(clickPoint);
744                         if (calcMin < min) {
745                             min = calcMin;
746                             closestObj = testObj;
747                         }
748                         calcMin = l.getP2().distance(clickPoint);
749                         if (calcMin < min) {
750                             min = calcMin;
751                             closestObj = testObj;
752                         }
753                     } else if (testObj instanceof myCircle) {
754                         Ellipse2D.Double c = ((myCircle) testObj)
755                                 .getSelectedEntity();
756                         Ellipse2D.Double in_c = myCircle
757                                 .getSmallerGraphicEntity(c);
758                         if ((c != null) && (c.intersects(x, y, 5, 5))
759                                 && (!in_c.intersects(x, y, 5, 5))) {
760                             clickOn = testObj;
761 
    nuevoColor=DXF_Color.getColor(myJColorChooser.col.getBackground());
762                                 _dxf._u._myTables.elementAt(i)._myLayers
763                                 .elementAt(j)._myEnt.elementAt(k)._color = nuevoColor;
764                             repaint();
765                             return;
766                         }
767                         calcMin = new Point2D.Double(c.getCenterX(),
768                                 c.getCenterY()).distance(clickPoint);
769                         if (calcMin < min) {
770                             min = calcMin;
771                             closestObj = testObj;
772                         }
773                     } else if (testObj instanceof myPolyline) {
774                         GeneralPath p = ((myPolyline) testObj)
775                                 .getSelectedEntity();
776                         if ((p != null) && (p.intersects(x, y, 5, 5))) {
777                             clickOn = testObj;
778                             return;
779                         }
780                         for (int a = 0; a < ((myPolyline) testObj)._myVertex
781                                 .size(); a++) {
782                             calcMin = new Point2D.Double(
783                                     ((myPolyline) testObj)._myVertex.elementAt(
784                                             a).X(),
785                                     ((myPolyline) testObj)._myVertex.elementAt(
786                                             a).Y()).distance(clickPoint);
787                             if (calcMin < min) {
788                                 min = calcMin;
789                                 closestObj = testObj;
790                             }
791                         }
792                     } else if (testObj instanceof myArc) {
793                         Arc2D.Double a = ((myArc) testObj).getSelectedEntity();
```

```
794                         if ((a != null) && (a.intersects(x, y, 5, 5))) {
795                             clickOn = testObj;
796
   nuevoColor=DXF_Color.getColor(myJColorChooser.col.getBackground());
797                                 _dxf._u._myTables.elementAt(i)._myLayers
798                              .elementAt(j)._myEnt.elementAt(k)._color = nuevoColor;
799                             repaint();
800                             return;
801                         }
802                         calcMin = new Point2D.Double(a.getCenterX(),
803                                 a.getCenterY()).distance(clickPoint);
804                         if (calcMin < min) {
805                             min = calcMin;
806                             closestObj = testObj;
807                         }
808                     } else if (testObj instanceof myBlockReference) {
809                         selEntityVar[0] = x;
810                         selEntityVar[1] = y;
811                         selEntityVar[2] = min;
812                         if (((myBlockReference) testObj)
813                                 .getSelectedEntity(selEntityVar)) {
814                             // clickOn=testObj;
815                             return;
816                         }
817                     }
818                 }
819             }
820         }
821         if (_dxf.proximitySelection) {
822             if (clickOn == null)
823                 clickOn = closestObj;
824         }
825     }
826
827     @Override
828     public void mouseDragged(MouseEvent e) {
829     /*  currPoint = e.getPoint();
830         _dxf.coordXY.setText("   X:  " + myCoord.javaToDXF_X(currPoint.x)
831                 + "  Y:  " + myCoord.javaToDXF_Y(currPoint.y));
832         if (this.selecting && (vectClickOn.size() > 0)) { // !=null)) {
833
834             if (this.editCircle != null) {
835                 Point2D.Double p = new Point2D.Double();
836                 p.x = myCoord.dxfToJava_X(editCircle._point.X());
837                 p.y = myCoord.dxfToJava_Y(editCircle._point.Y());
838                 this.editCircle._radius = p.distance(e.getPoint());
839             } else if (this.editArc != null) {
840                 Point2D.Double p = new Point2D.Double();
841                 p.x = myCoord.dxfToJava_X(editArc._point.X());
842                 p.y = myCoord.dxfToJava_Y(editArc._point.Y());
843                 this.editArc._radius = p.distance(e.getPoint());
844             } else {
845                 if (lastDrag == null)
846                     lastDrag = currPoint;
847                 double x = lastDrag.getX() - currPoint.getX();
848                 double y = lastDrag.getY() - currPoint.getY();
849                 for (int i = 0; i < vectClickOn.size(); i++) {
850                     clickOn = vectClickOn.elementAt(i);
851             //      if (clickOn != null)/* && (clickOn.selected)
852                  {
853                         if (clickOn instanceof myLine) {
854                             ((myLine) clickOn).translate(x, y);
```

```
855
856                         } else if (clickOn instanceof myCircle) {
857                             ((myCircle) clickOn).translate(x, y);
858                         } else if (clickOn instanceof myPolyline) {
859                             ((myPolyline) clickOn).translate(x, y);
860                         } else if (clickOn instanceof myArc) {
861                             ((myArc) clickOn).translate(x, y);
862                         } else if (clickOn instanceof myPoint) {
863                             ((myPoint) clickOn).translate(x, y);
864                         } else if (clickOn instanceof myText) {
865                             ((myText) clickOn).translate(x, y);
866                         } else if (clickOn instanceof mySolid) {
867                             ((mySolid) clickOn).translate(x, y);
868                         } else if (clickOn instanceof myBlockReference) {
869                             ((myBlockReference) clickOn).translate(x, y);
870                         }
871                     }
872                 }
873                 lastDrag = currPoint;
874             }
875         } else if (this.moving) {
876             if (lastDrag == null)
877                 lastDrag = lastClick;
878             if (e.getModifiersEx() == InputEvent.CTRL_DOWN_MASK) {
879                 this.moving = false;
880             } else {
881                 myCoord.decalageX += (currPoint.getX() - this.lastDrag.getX());
882                 myCoord.decalageY += (currPoint.getY() - this.lastDrag.getY());
883             }
884             lastDrag = currPoint;
885         } /*
886          * else if (this.xSelecting) {
887          *
888          * }
889          */
890
891     }
892
893     @Override
894     public void mouseReleased(MouseEvent e) {
895
896         if (this.drawingLine) {
897             if (e.getModifiersEx() == InputEvent.CTRL_DOWN_MASK) {
898                 this.drawingLine = false;
899             } else {
900                 myLine l = new myLine(new myPoint(new Point2D.Double(
901                         myCoord.javaToDXF_X(lastClick.getX()),
902                         myCoord.javaToDXF_Y(lastClick.getY()))), new myPoint(
903                         new Point2D.Double(myCoord.javaToDXF_X(e.getX()),
904                                 myCoord.javaToDXF_Y(e.getY()))),
905                         DXF_Color.getColor(_dxf._jcc.getColor()),
906                         _dxf._u.currLayer,
907                         (myLineType) _dxf._comboLineType.getSelectedItem(),
908                         _dxf._u.currThickness, 0);
909                 _dxf._u.currLayer._myEnt.addElement(l);
910                 _dxf.tree.addEntity(l);
911                 this.drawingLine = false;
912             }
913         } else if (this.drawingCircle) {
914             if (e.getModifiersEx() == InputEvent.CTRL_DOWN_MASK) {
915                 this.drawingCircle = false;
916             } else {
```

```
917                 myCircle c = new myCircle(new myPoint(new Point2D.Double(
918                         myCoord.javaToDXF_X(lastClick.getX()),
919                         myCoord.javaToDXF_Y(lastClick.getY()))),
920                         lastClick.distance(e.getPoint()) / myCoord.Ratio,
921                         (myLineType) _dxf._comboLineType.getSelectedItem(),
922                         DXF_Color.getColor(_dxf._jcc.getColor()),
923                         _dxf._u.currLayer, 0, _dxf._u.currThickness);
924                 _dxf._u.currLayer._myEnt.addElement(c);
925                 _dxf.tree.addEntity(c);
926                 this.drawingCircle = false;
927             }
928         } else if (this.drawingArc) {
929             if (e.getModifiersEx() == InputEvent.CTRL_DOWN_MASK) {
930                 this.drawingArc = false;
931                 this.drawingArcAngleStart = false;
932                 this.drawingArcAngleEnd = false;
933             } else {
934                 this.arcCenter = lastClick;
935                 this.arcRadius = e.getPoint();
936                 this.drawingArc = false;
937                 this.drawingArcAngleStart = true;
938             }
939         } else if (this.drawingArcAngleEnd) {
940             if (e.getModifiersEx() == InputEvent.CTRL_DOWN_MASK) {
941                 this.drawingArc = false;
942                 this.drawingArcAngleStart = false;
943                 this.drawingArcAngleEnd = false;
944             } else {
945                 double teta1 = Math.acos((arcStart.getX() - arcCenter.getX())
946                         / arcCenter.distance(arcStart));
947                 double teta2 = Math.acos((e.getPoint().getX() - arcCenter
948                         .getX()) / arcCenter.distance(e.getPoint()));
949 
950                 if ((arcStart.getY() < arcCenter.getY())
951                         && (e.getPoint().getY() > arcCenter.getY())) {// 1h ->
952                                                                       // 8h
953                     teta2 = (Math.PI * 2) - teta2;
954                 } else if ((arcStart.getY() > arcCenter.getY())
955                         && (e.getPoint().getY() > arcCenter.getY())) {// 8h ->
956                                                                       // 5h
957                     if (arcCenter.getX() < e.getX()) {
958                         teta1 = (Math.PI * 2) - teta1;
959                         teta2 = (Math.PI * 2) - teta2;
960                     } else {
961                         teta1 = (Math.PI * 2) - teta1;
962                         teta2 = (Math.PI * 2) + teta1;
963                     }
964                 } else if ((arcStart.getY() < arcCenter.getY())
965                         && (e.getPoint().getY() < arcCenter.getY())) {// 8h ->
966                                                                       // 1h
967                     if (arcStart.getX() < e.getX()) {
968                         teta1 = (Math.PI * 2) + teta1;
969                         teta2 = (Math.PI * 2) + teta2;
970                     } else {
971                     }
972                 } else if ((arcStart.getY() > arcCenter.getY())
973                         && (e.getPoint().getY() < arcCenter.getY())) {// 8h ->
974                                                                       // 1h
975                     double tmp = teta1;
976                     teta1 = (Math.PI * 2) + teta2;
977                     teta2 = (Math.PI * 2) - tmp;
978                 }
```

```
979
980                 myArc a = new myArc(Math.toDegrees(teta1 % (Math.PI * 2)),
981                         Math.toDegrees(teta2), new myPoint(new Point2D.Double(
982                                 myCoord.javaToDXF_X(arcCenter.getX()),
983                                 myCoord.javaToDXF_Y(arcCenter.getY()))),
984                         arcCenter.distance(arcRadius) / myCoord.Ratio,
985                         (myLineType) _dxf._comboLineType.getSelectedItem(),
986                         DXF_Color.getColor(_dxf._jcc.getColor()),
987                         _dxf._u.currLayer, 0, _dxf._u.currThickness);
988                 _dxf._u.currLayer._myEnt.addElement(a);
989                 _dxf.tree.addEntity(a);
990                 this.drawingArcAngleEnd = false;
991             }
992         } else if (this.moving) {
993             if ((e.getModifiersEx() != InputEvent.CTRL_DOWN_MASK)
994                     && (lastDrag != null)) {
995                 myCoord.decalageX += (e.getX() - this.lastDrag.getX());
996                 myCoord.decalageY += (e.getY() - this.lastDrag.getY());
997                 myHistory.saveHistory(true);
998                 myCanvas.clickOn = null;
999                 this.lastDrag = null;
1000                this.currPoint = null;
1001            }
1002            this.moving = false;
1003        } else if (this.zooming) {
1004            if (e.getModifiersEx() == InputEvent.CTRL_DOWN_MASK) {
1005                this.zooming = false;
1006                big.setStroke(currentStroke);
1007                zoomRect = null;
1008            } else {
1009                myCoord.Max = myCoord.javaToDXF_X(zoomRect.getWidth());
1010                //
    myCoord.decalageX*=-_dxf._u.lastView.getCenterX()-zoomRect.getCenterX();
1011                // myCoord.decalageY+=(this.getHeight()-zoomRect.getMaxY());
1012
1013                myCoord.resetRatio();
1014                myCoord.decalageX -= myCoord.dxfToJava_X(_dxf._u.lastView
1015                        .getCenterX() - zoomRect.getCenterX());
1016                myCoord.decalageY += myCoord.dxfToJava_Y(_dxf._u.lastView
1017                        .getCenterY() - zoomRect.getCenterY());
1018                // TODO --> 1
1019
1020                this.zooming = false;
1021                big.setStroke(currentStroke);
1022            }
1023        } else if (this.selecting) {// !=null)) {
1024            if (vectClickOn.size() > 0) {
1025                if ((e.getModifiersEx() == 0) || (e.getModifiersEx() == 64)) { // Bouton
1026                                                                               // 1
1027                                                                               // -
1028                                                                               // sans
1029                                                                               // Ctl
1030                    for (int i = 0; i < vectClickOn.size(); i++)
1031                        vectClickOn.elementAt(i).setSelected(false);
1032                    vectClickOn.removeAllElements();
1033
1034                }
1035
1036                if (this.editCircle != null) {
1037                    this.editCircle = null;
1038                }
1039                if (this.editArc != null) {
```

```
1040                    this.editArc = null;
1041                }
1042                if (editText == null) {
1043                    if (this.changingEnt != null) {
1044                        this.changingEnt.changing = false;
1045                        this.changingEnt = null;
1046                    }
1047                    clickOn = null;
1048                    lastDrag = null;
1049                    this.selecting = true;
1050                }
1051            }
1052
1053            _dxf.sel.setText(_dxf.defSelTxtA + vectClickOn.size() + _dxf.txtB);
1054            _dxf.clipB.setText(_dxf.defClipTxtA + clipBoard.size() + _dxf.txtB);
1055
1056        } else if (this.drawingEllipse) {
1057            double width = e.getX() - lastClick.getX();
1058            double height = e.getY() - lastClick.getY();
1059            double centerX = e.getX() + width / 2;
1060            myEllipse el = new myEllipse(new myPoint(new Point2D.Double(
1061                    centerX, e.getY() + height / 2)), new myPoint(
1062                    new Point2D.Double(centerX, e.getY())),
1063                    myCoord.dxfToJava_X(width) / myCoord.dxfToJava_Y(height),
1064                    0, 360, DXF_Color.getColor(_dxf._jcc.getColor()),
1065                    _dxf._u.currLayer, 0,
1066                    (myLineType) _dxf._comboLineType.getSelectedItem());
1067            _dxf._u.currLayer._myEnt.addElement(el);
1068            _dxf.tree.addEntity(el);
1069            this.drawingEllipse = false;
1070        } else if (this.xSelecting) { // try catch --> stackOverFlow je crois
1071            myEntity testObj;
1072            for (int i = 0; i < _dxf._u._myTables.size(); i++) {
1073                for (int j = 0; j < _dxf._u._myTables.elementAt(i)._myLayers
1074                        .size(); j++) {
1075                    for (int k = 0; k < _dxf._u._myTables.elementAt(i)._myLayers
1076                            .elementAt(j)._myEnt.size(); k++) {
1077                        testObj = _dxf._u._myTables.elementAt(i)._myLayers
1078                                .elementAt(j)._myEnt.elementAt(k);
1079                        if (testObj instanceof myPoint) {
1080                            Rectangle2D.Double p = ((myPoint) testObj)
1081                                    .getSelectedEntity();
1082                            if ((p != null) && (zoomRect.contains(p))) {
1083                                ((myPoint) testObj).setSelected(true);
1084                                vectClickOn.add(testObj);
1085                            }
1086                        } else if (testObj instanceof myText) {
1087                            Rectangle2D.Double r = ((myText) testObj)
1088                                    .getSelectedEntity();
1089                            if ((r != null) && (zoomRect.contains(r))) {
1090                                ((myText) testObj).setSelected(true);
1091                                vectClickOn.add(testObj);
1092                            }
1093                        } else if (testObj instanceof mySolid) {
1094                            GeneralPath p = ((mySolid) testObj)
1095                                    .getSelectedEntity();
1096                            if ((p != null)
1097                                    && (zoomRect.contains(p.getBounds2D()))) {
1098                                ((mySolid) testObj).setSelected(true);
1099                                vectClickOn.add(testObj);
1100                            }
1101                        } else if (testObj instanceof myLine) {
```

```
1102                            Line2D.Double l = ((myLine) testObj)
1103                                    .getSelectedEntity();
1104                            if ((l != null)
1105                                    && ((zoomRect.contains(l.x1, l.y1)) && (zoomRect
1106                                            .contains(l.x2, l.y2)))) {
1107                                ((myLine) testObj).setSelected(true);
1108                                vectClickOn.add(testObj);
1109                            }
1110                        } else if (testObj instanceof myCircle) {
1111                            Ellipse2D.Double c = ((myCircle) testObj)
1112                                    .getSelectedEntity();
1113                            if ((c != null)
1114                                    && (zoomRect.contains(c.getBounds2D()))) {
1115                                ((myCircle) testObj).setSelected(true);
1116                                vectClickOn.add(testObj);
1117                            }
1118                        } else if (testObj instanceof myPolyline) {
1119                            GeneralPath p = ((myPolyline) testObj)
1120                                    .getSelectedEntity();
1121                            if ((p != null)
1122                                    && (zoomRect.contains(p.getBounds2D()))) {
1123                                ((myPolyline) testObj).setSelected(true);
1124                                vectClickOn.add(testObj);
1125                            }
1126                        } else if (testObj instanceof myArc) {
1127                            Arc2D.Double a = ((myArc) testObj)
1128                                    .getSelectedEntity();
1129                            if ((a != null)
1130                                    && (zoomRect.contains(a.getBounds2D()))) {
1131                                ((myArc) testObj).setSelected(true);
1132                                vectClickOn.add(testObj);
1133                            }
1134                        } else if (testObj instanceof myBlockReference) {
1135                            ((myBlockReference) testObj)
1136                                    .setContainedEntitiesSelection(true,
1137                                            zoomRect, vectClickOn);
1138                        }
1139                    }
1140                }
1141            }
1142
1143            _dxf.sel.setText(_dxf.defSelTxtA + vectClickOn.size() + _dxf.txtB);
1144            _dxf.clipB.setText(_dxf.defClipTxtA + clipBoard.size() + _dxf.txtB);
1145
1146            xSelecting = false;
1147            zoomRect = null;
1148        }
1149
1150        _dxf.tree.updateSelection();
1151        repaint();
1152    }
1153
1154    @Override
1155    public void mouseClicked(MouseEvent e) {
1156            _dxf.tree.updateSelection();
1157        repaint();
1158    }
1159
1160    @Override
1161    public void mouseExited(MouseEvent e) {
1162        _dxf.coordXY.setText("");
1163    }
```

```
1164
1165    @Override
1166    public void mouseEntered(MouseEvent e) {
1167        _dxf.coordXY.setText("   X:  " + myCoord.javaToDXF_X(e.getX())
1168                + "  Y:  " + (myCoord.javaToDXF_Y(e.getY()) - 1));
1169    }
1170
1171    @Override
1172    public void mouseMoved(MouseEvent e) {
1173        currPoint = e.getPoint();
1174        _dxf.coordXY.setText("   X:  " + myCoord.javaToDXF_X(currPoint.x)
1175                + "  Y:  " + (myCoord.javaToDXF_Y(currPoint.y) - 1));
1176    }
1177
1178    public void stateChanged(ChangeEvent arg0) {
1179    }
1180
1181    @Override
1182    public void componentHidden(ComponentEvent arg0) {
1183    }
1184
1185    @Override
1186    public void componentResized(ComponentEvent arg0) {
1187        dim = getSize();
1188        myCoord.Max = dim.width;
1189        area = new Rectangle(dim);
1190        if (dim.width > 0 && dim.height > 0)
1191            bi = (BufferedImage) createImage(dim.width, dim.height);
1192        big = bi.createGraphics();
1193    }
1194
1195    @Override
1196    public void componentShown(ComponentEvent arg0) {
1197    }
1198
1199    @Override
1200    public void componentMoved(ComponentEvent arg0) {
1201    }
1202
1203    @Override
1204    public void keyPressed(KeyEvent arg0) {
1205    }
1206
1207    @Override
1208    public void keyReleased(KeyEvent arg0) {
1209    }
1210
1211    @Override
1212    public void keyTyped(KeyEvent ke) {
1213        // if ((ke.getModifiers()== KeyEvent.CTRL_MASK) && (ke.getKeyCode() ==
1214        // KeyEvent.VK_A))
1215        if ((_dxf._jcc.getSelectedIndex() == myJColorChooser.toolSel)) {
1216            int x = MouseInfo.getPointerInfo().getLocation().x;
1217            int y = MouseInfo.getPointerInfo().getLocation().y;
1218
1219            if ((drawingTxt) && (editText != null)) {
1220                if (firstTxt) {
1221                    editText.setVal("");
1222                    firstTxt = false;
1223                }
1224                if (ke.getKeyChar() == KeyEvent.VK_ENTER) {
1225                    drawingTxt = false;
```

```
1226                    if (this.changingEnt != null) {
1227                        this.changingEnt.changing = false;
1228                        this.changingEnt = null;
1229                    }
1230                    clickOn = null;
1231                } else if ((ke.getKeyChar() == KeyEvent.VK_DELETE)
1232                        || (ke.getKeyChar() == KeyEvent.VK_BACK_SPACE)) {
1233                    editText.delChar();
1234                } else
1235                    editText.appendVal(ke.getKeyChar());
1236            } else {
1237                if (vectClickOn.size() > 0) {
1238                    if ((KeyEvent.getKeyText(ke.getKeyChar())
1239                            .equalsIgnoreCase("Annuler"))
1240                            || (ke.getKeyChar() == 'c')
1241                            || (ke.getKeyChar() == 'C')) {
1242                        copySelectedObjectsToClipboard();
1243                    } else if (ke.getKeyChar() == 'x' || ke.getKeyChar() == 'X') {
1244                        cutSelectedObjectsToClipboard();
1245                    }
1246                }
1247                if (ke.getKeyChar() == 'p' || ke.getKeyChar() == 'P') {
1248                    if (this.clipBoard != null) {
1249                        pasteClipboardContents();
1250                    }
1251                } else if (ke.getKeyChar() == 'a' || ke.getKeyChar() == 'A') {
1252                    myEntity ent;
1253                    vectClickOn.removeAllElements();
1254                    for (int i = 0; i < _dxf._u._myTables.size(); i++) {
1255                        for (int j = 0; j < _dxf._u._myTables.elementAt(i)._myLayers
1256                                .size(); j++) {
1257                            for (int k = 0; k < _dxf._u._myTables.elementAt(i)._myLayers
1258                                    .elementAt(j)._myEnt.size(); k++) {
1259                                ent = _dxf._u._myTables.elementAt(i)._myLayers
1260                                        .elementAt(j)._myEnt.elementAt(k);
1261                                ent.setSelected(true);
1262                                vectClickOn.add(ent);
1263                            }
1264                        }
1265                    }
1266                } else if ((ke.getKeyChar() == 'i' || ke.getKeyChar() == 'I')) {
1267                    myBlock b = new myBlock(x, y, 0,
1268                            myNameGenerator.getBlockName(DXF_Loader.res
1269                                    .getString("myCanvas.1")),
1270                            new Vector<myEntity>(),
1271                            DXF_Color.getColor(_dxf._jcc.getColor()),
1272                            _dxf._u.currLayer, _dxf._u);
1273                    _dxf._u._myBlocks.add(b);
1274                    _dxf._u.currBlock = b;
1275                    myEntity ent;
1276                    for (int i = 0; i < _dxf._u._myTables.size(); i++) {
1277                        for (int j = 0; j < _dxf._u._myTables.elementAt(i)._myLayers
1278                                .size(); j++) {
1279                            int end = _dxf._u._myTables.elementAt(i)._myLayers
1280                                    .elementAt(j)._myEnt.size() - 1;
1281                            for (int k = end; k >= 0; k--) {
1282                                ent = _dxf._u._myTables.elementAt(i)._myLayers
1283                                        .elementAt(j)._myEnt.elementAt(k);
1284                                if (ent.selected) {
1285                                    ent.setSelected(false);
1286                                    b._myEnt.addElement(ent);
1287                                    _dxf._u._myTables.elementAt(i)._myLayers
```

```
                                        .elementAt(j)._myEnt
                                        .removeElementAt(k);
                                }
                            }
                        }
                    }
                    _dxf._u.currLayer._myEnt
                            .add(new myInsert(x, y, b._name, b,
                                    _dxf._u.currLayer, 0, DXF_Color
                                            .getColor(_dxf._jcc.getColor()),
                                    (myLineType) _dxf._comboLineType
                                            .getSelectedItem()));
                } else if ((ke.getKeyChar() == 'd' || ke.getKeyChar() == 'D')) {
                    myBlock b = new myBlock(x, y, 0,
                            myNameGenerator.getBlockName(DXF_Loader.res
                                    .getString("myCanvas.1")),
                            new Vector<myEntity>(),
                            DXF_Color.getColor(_dxf._jcc.getColor()),
                            _dxf._u.currLayer, _dxf._u);
                    _dxf._u._myBlocks.add(b);
                    _dxf._u.currBlock = b;
                    myEntity ent;
                    for (int i = 0; i < _dxf._u._myTables.size(); i++) {
                        for (int j = 0; j < _dxf._u._myTables.elementAt(i)._myLayers
                                .size(); j++) {
                            int end = _dxf._u._myTables.elementAt(i)._myLayers
                                    .elementAt(j)._myEnt.size() - 1;
                            for (int k = end; k >= 0; k--) {
                                ent = _dxf._u._myTables.elementAt(i)._myLayers
                                        .elementAt(j)._myEnt.elementAt(k);
                                if (ent.selected) {
                                    ent.setSelected(false);
                                    b._myEnt.addElement(ent);
                                    _dxf._u._myTables.elementAt(i)._myLayers
                                            .elementAt(j)._myEnt
                                            .removeElementAt(k);
                                }
                            }
                        }
                    }
                    _dxf._u.currLayer._myEnt
                            .add(new myDimension(0, myNameGenerator
                                    .getDimensionName(DXF_Loader.res
                                            .getString("myCanvas.2")), x, y, b,
                                    b._name, _dxf._u.currLayer, 0, DXF_Color
                                            .getColor(_dxf._jcc.getColor()),
                                    (myLineType) _dxf._comboLineType
                                            .getSelectedItem()));
                } else if ((ke.getKeyChar() == 'b' || ke.getKeyChar() == 'B')) {
                    myEntity ent;
                    for (int i = 0; i < _dxf._u._myTables.size(); i++) {
                        for (int j = 0; j < _dxf._u._myTables.elementAt(i)._myLayers
                                .size(); j++) {
                            int end = _dxf._u._myTables.elementAt(i)._myLayers
                                    .elementAt(j)._myEnt.size() - 1;
                            for (int k = end; k >= 0; k--) {
                                ent = _dxf._u._myTables.elementAt(i)._myLayers
                                        .elementAt(j)._myEnt.elementAt(k);
                                if (ent.selected) {
                                    ent.setSelected(false);
                                    if (_dxf._u.currBlock == null) {
                                        myBlock b = new myBlock(
```

```
1350                                                x,
1351                                                y,
1352                                                0,
1353                                                myNameGenerator
1354                                                        .getBlockName(DXF_Loader.res
1355                                                                .getString("myCanvas.1")),
1356                                                new Vector<myEntity>(),
1357                                                DXF_Color.getColor(_dxf._jcc
1358                                                        .getColor()),
1359                                                _dxf._u.currLayer, _dxf._u);
1360                                        _dxf._u._myBlocks.add(b);
1361                                        _dxf._u.currBlock = b;
1362                                    }
1363                                    _dxf._u.currBlock._myEnt.addElement(ent);
1364                                    _dxf._u._myTables.elementAt(i)._myLayers
1365                                            .elementAt(j)._myEnt
1366                                            .removeElementAt(k);
1367                                }
1368                            }
1369                        }
1370                    }
1371                    vectClickOn.removeAllElements();
1372                } else if ((ke.getKeyChar() == KeyEvent.VK_DELETE)
1373                        || (KeyEvent.getKeyText(ke.getKeyChar())
1374                                .equalsIgnoreCase("Supprimer"))) {
1375                    for (int i = 0; i < _dxf._u._myTables.size(); i++) {
1376                        for (int j = 0; j < _dxf._u._myTables.elementAt(i)._myLayers
1377                                .size(); j++) {
1378                            int end = _dxf._u._myTables.elementAt(i)._myLayers
1379                                    .elementAt(j)._myEnt.size() - 1;
1380                            for (int k = end; k >= 0; k--) {
1381                                clickOn = _dxf._u._myTables.elementAt(i)._myLayers
1382                                        .elementAt(j)._myEnt.elementAt(k);
1383                                if (clickOn.selected) {
1384                                    _dxf._u._myTables.elementAt(i)._myLayers
1385                                            .elementAt(j)._myEnt
1386                                            .removeElementAt(k);
1387                                }
1388                            }
1389                        }
1390                    }
1391                } else if (ke.getKeyChar() == KeyEvent.VK_ESCAPE) {
1392                    for (int i = 0; i < vectClickOn.size(); i++)
1393                        vectClickOn.elementAt(i).setSelected(false);
1394                    vectClickOn.removeAllElements();
1395                }
1396            }
1397
1398            _dxf.sel.setText(_dxf.defSelTxtA + vectClickOn.size() + _dxf.txtB);
1399            _dxf.clipB.setText(_dxf.defClipTxtA + clipBoard.size() + _dxf.txtB);
1400
1401        }/* else if (_dxf._typeOutil.getSelectedIndex() == myToolBar.toolSelX) {
1402            if (ke.getKeyChar() == KeyEvent.VK_ESCAPE) {
1403                for (int i = 0; i < vectClickOn.size(); i++)
1404                    vectClickOn.elementAt(i).setSelected(false);
1405            }
1406
1407            _dxf.sel.setText(_dxf.defSelTxtA + vectClickOn.size() + _dxf.txtB);
1408            _dxf.clipB.setText(_dxf.defClipTxtA + clipBoard.size() + _dxf.txtB);
1409
1410        } */else if ((drawingTxt) && (editText != null)) {
1411            if (firstTxt) {
```

```
1412                editText.setVal("");
1413                firstTxt = false;
1414            }
1415            if (ke.getKeyChar() == KeyEvent.VK_ENTER) {
1416                drawingTxt = false;
1417            } else if ((ke.getKeyChar() == KeyEvent.VK_DELETE)
1418                    || (ke.getKeyChar() == KeyEvent.VK_BACK_SPACE)) {
1419                editText.delChar();
1420            } else
1421                editText.appendVal(ke.getKeyChar());
1422        }
1423        _dxf.tree.createNodes();
1424    }
1425
1426    private void pasteClipboardContents() {
1427        myEntity ent;
1428        double x = 0, y = 0;
1429        if (origPoint != null) {
1430            x = origPoint.getX() - MouseInfo.getPointerInfo().getLocation().x;
1431            y = origPoint.getY() - MouseInfo.getPointerInfo().getLocation().y;
1432        }
1433        for (int i = 0; i < clipBoard.size(); i++) {
1434            ent = clipBoard.elementAt(i);
1435            if (ent instanceof myLine) {
1436                myLine l = new myLine((myLine) ent);
1437                l.translate(x, y);
1438                _dxf._u.currLayer._myEnt.add(l);
1439            } else if (ent instanceof myCircle) {
1440                myCircle c = new myCircle((myCircle) ent);
1441                c.translate(x, y);
1442                _dxf._u.currLayer._myEnt.add(c);
1443            } else if (ent instanceof myPolyline) {
1444                myPolyline p = new myPolyline((myPolyline) ent);
1445                p.translate(x, y);
1446                _dxf._u.currLayer._myEnt.add(p);
1447            } else if (ent instanceof myArc) {
1448                myArc a = new myArc((myArc) ent);
1449                a.translate(x, y);
1450                _dxf._u.currLayer._myEnt.add(a);
1451            } else if (ent instanceof myPoint) {
1452                myPoint m = new myPoint((myPoint) ent);
1453                m.translate(x, y);
1454                _dxf._u.currLayer._myEnt.add(m);
1455            } else if (ent instanceof myText) {
1456                myText m = new myText((myText) ent);
1457                m.translate(x, y);
1458                _dxf._u.currLayer._myEnt.add(m);
1459            } else if (ent instanceof myTrace) {
1460                myTrace m = new myTrace((myTrace) ent);
1461                m.translate(x, y);
1462                _dxf._u.currLayer._myEnt.add(m);
1463            } else if (ent instanceof mySolid) {
1464                mySolid m = new mySolid((mySolid) ent);
1465                m.translate(x, y);
1466                _dxf._u.currLayer._myEnt.add(m);
1467            } else if (ent instanceof myBlockReference) {
1468                pasteTx(((myBlockReference) ent)._refBlock._myEnt, x, y);
1469            }
1470        }
1471    }
1472
1473    public void pasteTx(Vector<myEntity> vmy, double x, double y) {
```

```
1474        myEntity my;
1475        for (int a = 0; a < vmy.size(); a++) {
1476            my = vmy.elementAt(a);
1477            if (my instanceof myLine) {
1478                _dxf._u.currLayer._myEnt.add((new myLine((myLine) my)));
1479                _dxf._u.currLayer._myEnt.lastElement().translate(x, y);
1480                _dxf._u.currLayer._myEnt.lastElement()._refLayer = _dxf._u.currLayer;
1481            } else if (my instanceof myCircle) {
1482                _dxf._u.currLayer._myEnt.add((new myCircle((myCircle) my)));
1483                _dxf._u.currLayer._myEnt.lastElement().translate(x, y);
1484                _dxf._u.currLayer._myEnt.lastElement()._refLayer = _dxf._u.currLayer;
1485            } else if (my instanceof myPolyline) {
1486                _dxf._u.currLayer._myEnt.add((new myPolyline((myPolyline) my)));
1487                _dxf._u.currLayer._myEnt.lastElement().translate(x, y);
1488                _dxf._u.currLayer._myEnt.lastElement()._refLayer = _dxf._u.currLayer;
1489            } else if (my instanceof myArc) {
1490                _dxf._u.currLayer._myEnt.add((new myArc((myArc) my)));
1491                _dxf._u.currLayer._myEnt.lastElement().translate(x, y);
1492                _dxf._u.currLayer._myEnt.lastElement()._refLayer = _dxf._u.currLayer;
1493            } else if (my instanceof myText) {
1494                _dxf._u.currLayer._myEnt.add((new myText((myText) my)));
1495                _dxf._u.currLayer._myEnt.lastElement().translate(x, y);
1496                _dxf._u.currLayer._myEnt.lastElement()._refLayer = _dxf._u.currLayer;
1497            } else if (my instanceof mySolid) {
1498                _dxf._u.currLayer._myEnt.add((new mySolid((mySolid) my)));
1499                _dxf._u.currLayer._myEnt.lastElement().translate(x, y);
1500                _dxf._u.currLayer._myEnt.lastElement()._refLayer = _dxf._u.currLayer;
1501            } else if (my instanceof myPoint) {
1502                _dxf._u.currLayer._myEnt.add((new myPoint((myPoint) my)));
1503                _dxf._u.currLayer._myEnt.lastElement().translate(x, y);
1504                _dxf._u.currLayer._myEnt.lastElement()._refLayer = _dxf._u.currLayer;
1505            } else if (my instanceof myBlockReference) {
1506                pasteTx(((myBlockReference) my)._refBlock._myEnt, x, y); // Catch
1507                                                                         // overflow
1508                                                                         // ?
1509            }
1510        }
1511    }
1512
1513    public void cutSelectedObjectsToClipboard() {
1514        copySelectedObjectsToClipboard();
1515        origPoint = MouseInfo.getPointerInfo().getLocation();
1516        myEntity test;
1517        for (int i = 0; i < _dxf._u._myTables.size(); i++) {
1518            for (int j = 0; j < _dxf._u._myTables.elementAt(i)._myLayers.size(); j++) {
1519                int end = _dxf._u._myTables.elementAt(i)._myLayers.elementAt(j)._myEnt
1520                        .size() - 1;
1521                for (int k = end; k >= 0; k--) {
1522                    test = _dxf._u._myTables.elementAt(i)._myLayers
1523                            .elementAt(j)._myEnt.elementAt(k);
1524                    if (test.selected) {
1525                        _dxf._u._myTables.elementAt(i)._myLayers.elementAt(j)._myEnt
1526                                .removeElementAt(k);
1527                    }
1528                }
1529            }
1530        }
1531        vectClickOn.removeAllElements();
1532    }
1533
1534    public void copySelectedObjectsToClipboard() {
1535        origPoint = null;
```

```
1536        this.clipBoard.removeAllElements();
1537        for (int i = 0; i < vectClickOn.size(); i++) {
1538            this.clipBoard.addElement(vectClickOn.elementAt(i));
1539        }
1540
1541    }
1542
1543    public static void myCanvas() {
1544        // TODO Auto-generated method stub
1545
1546    }
1547
1548 }
1549
1550
```

```java
/*-------------------------------------------------------------------------------------
Copyright 2007, Stephan Soulard and Edouard Vanhauwaert.
Copyright 2014, Celeste Gabriela Guagliano.

This file was originaly part of DXF project and then modified by
Celeste Gabriela Guagliano for DXF2Machine project.

DXF2Machine is free software: you can redistribute it and/or modify it under the terms of
the GNU General Public License as published by the Free Software Foundation, either
version 2 of the License.

DXF2Machine is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with DXF2Machine.
If not, see <http://www.gnu.org/licenses/>.
  -------------------------------------------------------------------------------------*/

/*
 * Initials     Name
 * ----------------------------------
 * CeGu         Celeste Guagliano.
 */

/*
 * modification history (new versions first)
 * -----------------------------------------------------------
 * 20150113 v0.0.2 CeGu improve actionPerformed() method
 * 20150113 v0.0.2 CeGu improve myJColorChooser() method
 * 20140428 v0.0.1 CeGu fork from DXF project

*/
package myDXF.Graphics;

import java.awt.BorderLayout;

/**
 * This class generate a Color Chooser.
 * @author: Stephan Soulard, Edouard Vanhauwaert.
 * @version: 13/01/15 by Celeste Guagliano
 *
 */
public class myJColorChooser extends JPanel implements ActionListener {
    public int selectedIndex = 0;


    public static final int toolSel = 6;
    public static final int toolNone = 0;

    private static final long serialVersionUID = 1L;
    public static JButton col;
    public static JButton colLayer;
    public static Hashtable colores=new Hashtable();
    public static JLabel color= new JLabel();
    public static JButton rojo=new JButton();
    public static JButton verde=new JButton();
    public static JButton cian=new JButton();
    public static JButton azul=new JButton();
    public static JButton nada=new JButton();
    public int indice=0;
    public myJColorChooser() {
```

```
81          colores.put(1,"Origen");
82          colores.put(3, "Taladrado");
83          colores.put(4, "Grabado");
84          colores.put(5, "Contorno");
85          colores.put(0,"Ningun Rasgo");
86          this.setLayout(new BorderLayout());
87          this.setPreferredSize(new Dimension(200, 20));
88          JPanel p = new JPanel(new GridLayout(0, 10));
89          color.setText((String)colores.get(0));
90          rojo.setCursor(Cursor
91                  .getPredefinedCursor(Cursor.HAND_CURSOR));
92          rojo.setPreferredSize(new Dimension(30,15));
93          rojo.addActionListener(this);
94          rojo.setActionCommand("selection");
95          rojo.setBackground(DXF_Color.ColorMap[1]);
96          rojo.setToolTipText((String) colores.get(1));
97          p.add(rojo);
98          rojo.addActionListener(new ActionListener() {
99              @Override
100             public void actionPerformed(ActionEvent e) {
101                 if (e.getSource() == rojo) {
102                     color.setText((String) colores.get(1));
103             }}}
104         );
105         verde.setCursor(Cursor
106                 .getPredefinedCursor(Cursor.HAND_CURSOR));
107         verde.setPreferredSize(new Dimension(8, 8));
108         verde.addActionListener(this);
109         verde.setActionCommand("selection");
110         verde.setBackground(DXF_Color.ColorMap[3]);
111         verde.setToolTipText((String) colores.get(3));
112         p.add(verde);
113         verde.addActionListener(new ActionListener() {
114             @Override
115             public void actionPerformed(ActionEvent e) {
116                 if (e.getSource() == verde) {
117                     color.setText((String) colores.get(3));
118             }}}
119         );
120         cian.setCursor(Cursor
121                 .getPredefinedCursor(Cursor.HAND_CURSOR));
122         cian.setPreferredSize(new Dimension(8, 8));
123         cian.addActionListener(this);
124         cian.setActionCommand("selection");
125         cian.setBackground(DXF_Color.ColorMap[4]);
126         cian.setToolTipText((String) colores.get(4));
127         p.add(cian);
128         cian.addActionListener(new ActionListener() {
129             @Override
130             public void actionPerformed(ActionEvent e) {
131                 if (e.getSource() == cian) {
132                     color.setText((String) colores.get(4));
133             }}}
134         );
135         azul.setCursor(Cursor
136                 .getPredefinedCursor(Cursor.HAND_CURSOR));
137         azul.setPreferredSize(new Dimension(8, 8));
138         azul.addActionListener(this);
139         azul.setActionCommand("selection");
140         azul.setBackground(DXF_Color.ColorMap[5]);
141         azul.setToolTipText((String) colores.get(5));
142         p.add(azul);
```

```
143         azul.addActionListener(new ActionListener() {
144             @Override
145             public void actionPerformed(ActionEvent e) {
146                 if (e.getSource() == azul) {
147                     color.setText((String) colores.get(5));
148             }}}
149         );
150 
151         nada.setCursor(Cursor
152                 .getPredefinedCursor(Cursor.HAND_CURSOR));
153         nada.setPreferredSize(new Dimension(8, 8));
154         nada.addActionListener(this);
155         nada.setActionCommand("nada");
156         nada.setBackground(DXF_Color.ColorMap[0]);
157         nada.setToolTipText((String) colores.get(0));
158         p.add(nada);
159         nada.addActionListener(new ActionListener() {
160             @Override
161             public void actionPerformed(ActionEvent e) {
162                 if (e.getSource() == nada) {
163                     color.setText((String) colores.get(0));
164             }}}
165         );
166 
167 /*      for (int i = 3; i < 6; i++) {
168             tmpJButton = new JButton();
169             tmpJButton.setActionCommand("Selection");
170             tmpJButton.setCursor(Cursor
171                     .getPredefinedCursor(Cursor.HAND_CURSOR));
172             tmpJButton.setPreferredSize(new Dimension(8, 8));
173             tmpJButton.addActionListener(this);
174             tmpJButton.setBackground(DXF_Color.ColorMap[i]);
175             tmpJButton.setToolTipText((String) colores.get(i));
176             p.add(tmpJButton);
177         }*/
178         this.add(p, BorderLayout.NORTH);
179         this.setPreferredSize(new Dimension(200, 20));
180 
181         this.setMinimumSize(new Dimension(200, 20));
182 
183         JPanel p_current = new JPanel();
184 
185     /*  p_current
186                 .add(new JLabel(DXF_Loader.res.getString("myJColorChooser.0")));
187 
188        colLayer = new JButton();
189         colLayer.setMinimumSize(new Dimension(50, 9));
190         colLayer.setPreferredSize(new Dimension(50, 9));
191         colLayer.addMouseListener(new MouseAdapter() {
192             @Override
193             public void mouseReleased(final MouseEvent e) {
194                 col.setBackground(colLayer.getBackground());
195             }
196         });
197         colLayer.setBackground(DXF_Color.getDefaultColor());
198         colLayer.setEnabled(true);
199         colLayer.setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
200         colLayer.setToolTipText("0");
201 
202         p_current.add(colLayer);
203 */
204         //color=new JLabel((String) colores.get(0));
```

```
205         p_current.add(color);
206         col = new JButton("");
207         col.setMinimumSize(new Dimension(50, 20));
208         col.setPreferredSize(new Dimension(50, 20));
209         if (DXF_Loader._mc == null || myCanvas._dxf == null
210                 || myCanvas._dxf._u == null
211                 || myCanvas._dxf._u.currLayer == null)
212             col.setBackground(DXF_Color.getDefaultColor());
213         else
214             col.setBackground(DXF_Color
215                     .getColor(myCanvas._dxf._u.currLayer._color));
216 
217         col.setEnabled(false);
218         col.setToolTipText(DXF_Loader.res.getString("myJColorChooser.4"));
219 
220         p_current.add(col);
221         this.add(p_current, BorderLayout.EAST);
222 
223     }
224 
225     public Color getColor() {
226         return col.getBackground();
227     }
228 
229     @Override
230     public void actionPerformed(ActionEvent a) {
231         col.setBackground(((JButton) a.getSource()).getBackground());
232 
233         setCursor(Cursor.getDefaultCursor());
234         DXF_Loader._mc.setCursor(Cursor
235                 .getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
236         boolean s = true;
237 
238 
239 
240             if (a.getActionCommand() == "nada") {
241                 DXF_Loader._mc.setCursor(Cursor
242                         .getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
243                 selectedIndex = toolNone;
244                 s = true;
245             }
246                 if (a.getActionCommand() == "selection") {
247                 DXF_Loader._mc.setCursor(Cursor
248                         .getPredefinedCursor(Cursor.HAND_CURSOR));
249                 selectedIndex = toolSel;
250                 s = false;
251             }
252 
253 
254 
255         reset(s);
256 
257     }
258 
259     public void reset(boolean s) {
260         if (s) {
261             for (int i = 0; i < DXF_Loader._mc.vectClickOn.size(); i++)
262                 DXF_Loader._mc.vectClickOn.elementAt(i).setSelected(false);
263             DXF_Loader._mc.vectClickOn.removeAllElements();
264         }
265 
266         DXF_Loader._mc.selecting = false;
```

```
267         DXF_Loader._mc.moving = false;
268         DXF_Loader._mc.zooming = false;
269         DXF_Loader._mc.drawingCircle = false;
270         DXF_Loader._mc.drawingPolyLineStart = false;
271         DXF_Loader._mc.drawingPolyLineEnd = false;
272         DXF_Loader._mc.drawingArc = false;
273         DXF_Loader._mc.drawingArcAngleStart = false;
274         DXF_Loader._mc.drawingArcAngleEnd = false;
275         DXF_Loader._mc.drawingEllipse = false;
276         DXF_Loader._mc.drawingTrace = false;
277         DXF_Loader._mc.drawingTxt = false;
278         DXF_Loader._mc.drawingSolid = false;
279 
280         myCanvas._dxf.sel.setText(myCanvas._dxf.defSelTxtA
281                 + DXF_Loader._mc.vectClickOn.size() + myCanvas._dxf.txtB);
282         myCanvas._dxf.clipB.setText(myCanvas._dxf.defClipTxtA
283                 + DXF_Loader._mc.clipBoard.size() + myCanvas._dxf.txtB);
284     }
285 
286     public int getSelectedIndex() {
287         return selectedIndex;
288     }
289 
290 }
291 
```

```java
/*-------------------------------------------------------------------------------------
Copyright 2007, Stephan Soulard and Edouard Vanhauwaert.
Copyright 2014, Celeste Gabriela Guagliano.

This file was originaly part of DXF project and then modified by
Celeste Gabriela Guagliano for DXF2Machine project.

DXF2Machine is free software: you can redistribute it and/or modify it under the terms of
the GNU General Public License as published by the Free Software Foundation, either
version 2 of the License.

DXF2Machine is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with DXF2Machine.
If not, see <http://www.gnu.org/licenses/>.
  -------------------------------------------------------------------------------------*/

/*
 * Initials     Name
 * ----------------------------------
 * CeGu         Celeste Guagliano.
 */

/*
 * modification history (new versions first)
 * -----------------------------------------------------------
 * 20150113 v0.0.2 CeGu modified myToolbar class
 * 20140428 v0.0.1 CeGu fork from DXF project

*/
package myDXF.Graphics;

import java.awt.Cursor;

public class myToolBar extends JToolBar implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent arg0) {
        // TODO Auto-generated method stub

    }


}

```

Bibliografía

[1] Sandvik Coromant. Modern metal cutting: a practical handbook. 1994.

[2] Wikipedia. Mecanizado — wikipedia, la enciclopedia libre, 2015. URL https://es.wikipedia.org/w/index.php?title=Mecanizado&oldid=81419150.

[3] Wikipedia. Fresadora — wikipedia, la enciclopedia libre, 2015. URL https://es.wikipedia.org/w/index.php?title=Fresadora&oldid=83470909.

[4] Wikipedia. Cad/cam — wikipedia, la enciclopedia libre, 2015. URL https://es.wikipedia.org/w/index.php?title=CAD/CAM&oldid=79563455.

[5] *PowerMill 2012 R2:Getting Started.*

[6] *NX 9 Tutorial.*

[7] *Manual MasterCAM X2.*

[8] *CATIA V5-6R2014 For Beginners.*

[9] *BobCAD-CAM V24.*

[10] *CamWorks 2014: Mill and Turn Tutorial.*

[11] *HeeksCNC.*

[12] *EMCO Pc Mill 55: Manual de instalación.*

[13] *EMCO WinNC: Sinumerik 810/820M.*

[14] *VF/HS:Programming workbook.*

[15] Autodesk INC. Dxf reference. 2007.

[16] Andres Juarez. Java: Programación orientada a objetos. 2014.

[17] Leonardo Gassman. Java para programadores objetosos. 2008.