INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

"Teleoperación de robots CNC"

T E S I S

Que para obtener el grado de

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

Presenta

ING. ISAÍ FARARONI RAMÍREZ

Director de tesis: Dr. Juan Luís Díaz de León Santiago

MÉXICO D.F. NOVIEMBRE 2005

Índice general

Índice de figuras

Índice de tablas

Resumen

Teleoperación de robots CNC

Los robots controlados numéricamente son ampliamente usados en los procesos industriales, educativos y de investigación. Muchos de ellos fueron desarrollados antes del surgimiento de Internet, por lo que la operación se realiza localmente y en equipos de bajo poder de cómputo. Esto obliga al operador a estar físicamente junto al equipo para poder usarlo.

En este trabajo se presenta el desarrollo e implementación de una arquitectura que facilita el desarrollo de sistemas de teleoperación de robots controlados numéricamente.

La arquitectura se compone de dos elementos: Control CNC y Teleoperación CNC, integrados en un Framework.

El componente **Control CNC** ofrece los métodos requeridos para interpretar programas de comandos RS274-D (G-Codes) y controlar el hardware. La implementación se realiza extendiendo las clases correspondientes al *driver* para manipular el robot. Tiene asociado un driver para controlar el robot y se conecta físicamente a él por medio del puerto paralelo.

El componente **Teleoperación CNC**, implementa un protocolo de comunicación basado en el modelo cliente-servidor. El servidor se ejecuta en la computadora que tiene conectado físicamente al robot, el protocolo recibe las instrucciones y las envía al componente Control CNC. El cliente es el componente diseñado para interactuar con los usuarios desde consola o editor de comandos, se ejecuta en una computadora distinta conectada a la red de Internet.

Es una arquitectura orientada a objetos en la que se han incorporado conceptos de patrones de diseño, para construir una arquitectura estable, estandarizada y abierta, que garantice el crecimiento gradual y consistente del modelo.

El resultado del proyecto es la construcción de un ambiente integrado de software para la teleoperación del robot MAXNC-10.

Abstract

CNC robots Teleoperation

The Numerically controlled robots are widely used in industrial, educational and of research processes. Many of them were developed before the arise of the Internet, this was the reason of why the operation is made locally and in low computing power equipment. This forces the operator to be physically next to the equipment to be able to use it.

In this work, we show the development and implementation of architecture that facilitates the development of systems for the teleoperation of numerically controlled robots.

The architecture is made up of two elements: "*Control CNC*" and "*Teleoperación CNC*", integrated in a Framework.

The component "*Control CNC*" offers the required methods to interpret programs in RS274-D commandos (G-Codes) and to control the hardware. The implementation is made extending the classes of the driver that manages the robot. It has associated a driver to control the robot and it is physically connected to it through a parallel port.

The component "*Teleoperación CNC*", implements a communication protocol based on the client-server model. The server executes itself in a computer that is physically connected to the robot; the protocol receives the instructions and sends them to the component "*Control CNC*". The client is the component designed to interact with the users from a console or from the command editor, which is executed in another computer connected to Internet.

The proposed Framework is an object oriented architecture in which the concepts of patterns design have been used, to construct a stable architecture, standardized and open, that guarantees the gradual and consistent growth of the model.

The final result of the project was the implementation of integrated software for the teleoperation of the MAXNC-10 robot.

Capítulo 1 Introducción

1.1 Objetivos

1.1.1 Objetivo general

Diseñar una arquitectura para el control y teleoperación de robots CNC, compuesto por un Framework para el desarrollo de aplicaciones de control de robots CNC y un protocolo de comunicación para la teleoperación.

1.1.2 Objetivos específicos

1. Diseñar un Framework para el desarrollo de aplicaciones de control de robots de control numérico.
2. Diseñar un componente para el control de robots CNC, que interprete programas RS274-D.
3. Diseñar un protocolo de comunicación para la teleoperación de robots CNC.
4. Desarrollar un controlador para el robot MAXNC-10 a partir del Framework.
5. Desarrollar una aplicación de escritorio que permita la edición, simulación y ejecución de comandos RS274-D, construida usando el Framework.

1.2 Motivación

Este proyecto surge en el GRAI (Grupo de Robótica y Análisis de Imágenes) del Centro de Investigación en Computación, como una necesidad para operar remotamente al robot MAXNC-10. Inicialmente se detectaron dos limitaciones que dificultaban su uso: primero, el software de control del robot solo puede ser instalado en la plataforma DOS, segundo, la operación no se puede realizar remotamente.

Los robots CNC juegan un papel importante en el desarrollo industrial, en el ámbito educativo y en las áreas de investigación.

Los modernos ambientes de fabricación hacen uso de recursos de diversos sistemas especializados para la realización de un producto: manipuladores robóticos, sistemas de transporte, organizadores, estaciones de trabajo CAD/CAM

y sistemas de diseño. En el ámbito educativo son ampliamente usados para la fabricación de piezas y diseño de prototipos. Muchas universidades no tienen acceso a este tipo de robots, por el alto costo de inversión. En las áreas de investigación participan activamente, para la fabricación de piezas de otros robots, por ejemplo, algunas piezas de los robots móviles Opportunity y Spirit que fueron enviados a la exploración de Marte en 2003, se fabricaron con ayuda de los robots CNC Haas VF-2 y VF5 [16].

El surgimiento y popularidad de la red Internet ha permitido la creación de nuevos paradigmas de trabajo. En la industria, se presenta como una alternativa que permite comunicar clientes, proveedores y *herramientas* de manufactura. En el ámbito educativo, abre la posibilidad de tener acceso a equipo especializado, compartiendo herramientas entre académicos y centros de investigación, elimina barreras de tiempo y distancia. Se pueden desarrollar sistemas de operación, monitoreo, registro y diagnóstico remoto.

La integración de los robots CNC a estos nuevos paradigmas de trabajo, requiere de una infraestructura de comunicación y control, diseñado ex profeso para la operación de este tipo de robots.

Control Numérico (NC)
El término "control numérico" se refiere a que las órdenes dadas al robot se indican mediante códigos numéricos, donde la posición es la principal variable a controlar. Los valores numéricos representan las posiciones esperadas de las herramientas o tienen una representación simbólica de funciones secundarias [28].

Control Numérico Computarizado (CNC)
Se refiere específicamente al control *por computadora* de los robots NC, con el propósito de fabricar repetidamente partes complejas (líneas, círculos, figuras tridimensionales) en metal y otros materiales, usando un programa escrito en una notación numérica[38].

Las máquinas de control numérico surgen en el año de 1952, como una necesidad de la US Air Force para la fabricación de piezas de precisión para un nuevo modelo de aviones supersónicos de la época. John T. Parsons de la Parsons Corporation (Traverse City, Michigan) identificó piezas que no podrían ser fabricadas con los métodos tradicionales, esto lo llevó a perfeccionar un método que había iniciado en 1947, para el año de 1952 registra la patente [45] de la primera máquina controlada numéricamente. Este invento fué considerado como el principio de la segunda revolución industrial y el advenimiento de una era en que el control de las máquinas pasaría de un trabajo artesanal a una ciencia exacta. La primera máquina controlada numéricamente también fué la primera que combinó el uso de servo-mecanismos con las computadoras digitales [32].

En el año de 1949 Pearsons firma un contrato con la US Air Force para diseñar y construir una fresadora que usara servo-mecanismos controlados por tarjetas perforadas o cintas de casete, para fabricar componentes de los aviones. En este

proyecto participa el Servomechanisms Laboratory del MIT dirigido por el Profesor Gordon Brown [48] con la tecnología de precisión, sin embargo el laboratorio toma un camino distinto al que Pearson había planteado originalmente, ellos diseñaron un sistema tridimensional de servos controlados digitalmente, en una fresadora convencional. A principios de Marzo de 1952, se presenta un prototipo funcional, validando la esencia de la invención de Pearson.

A finales de 1955 la US Air Force autoriza la compra de 105 máquinas de control numérico, iniciando su aplicación comercial. Sin embargo el tiempo requerido para preparar los programas manualmente, era muy largo y el proceso muy complejo. Durante los años 1956-59, Douglas Ross desarrolló un lenguaje para solucionar el problema, Automatically Programmed Tool (APT) [49][15]. Este lenguaje se convirtió en 1974 en el estándar americano para programar máquinas de control numérico y en el estándar internacional en 1978.

En los años 1950's surge la idea de integrar el proceso de fabricación con los otros procesos que participaban en la realización del producto y se crea el concepto: "Computer Integrated Machining System" [36]. Este sistema integrado parte desde el concepto inicial, diseño del producto, programación y control de su producción a la producción misma (Figura 1).

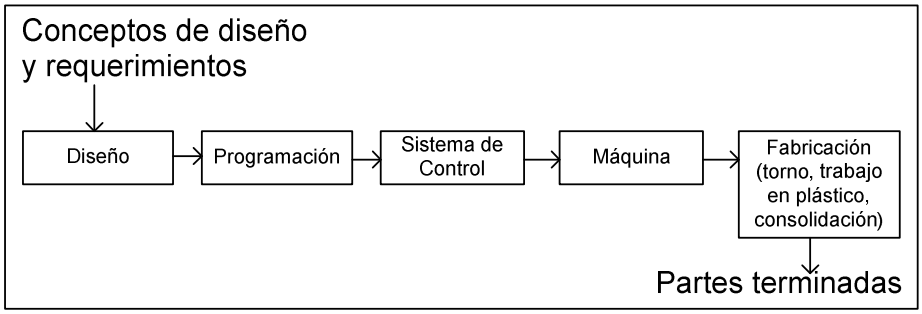


Figura 1 Sistema de fabricación integrada computacionalmente.

Con la disminución del tamaño y costo de las computadoras surge en los 1960's el Control Numérico Computarizado (CNC) [34] para equipos con capacidad de cambiar automáticamente sus herramientas, forman parte de centros de fabricación donde se integran con sistemas de manejo automático de materiales y piezas [39].

El estándar recomendado actualmente para escribir programas para las máquinas de control numérico controlados por computadora es el RS274-D conocido comúnmente como G-Code, fué desarrollado por el EIA (Electronic Industry Association) alrededor de los 1960's y aprobado en febrero de 1980[17] .

Las funciones de control de un robot CNC se implementan tradicionalmente en microcontroladores, los equipos más modernos implementan estas funciones de control en computadoras vía software y se comunican a los dispositivos por medio

de las interfaces serial o paralelo. El software de control está diseñado para ejecutarse en sistemas operativos antiguos.

1.3 Planteamiento del problema

Se han tomando en cuenta tres aspectos importantes, que revelan la necesidad de operar robots CNC de forma remota.

Contexto del problema
Los robots CNC fueron desarrolladas tomando en cuenta barreras tecnológicas, que hoy no existen. El bajo costo de fabricación del hardware ha permitido que el software de control se implemente con mayor frecuencia en computadoras personales. Las redes de comunicación son cada vez más confiables y veloces.

Fueron pensados en que su única función sería fabricar piezas, sin recibir o proporcionar retroalimentación a su entorno. La gran mayoría fueron fabricadas antes del crecimiento de la red de Internet y en equipos de bajo poder de procesamiento.

Barreras de migración
A pesar de que las condiciones tecnológicas han cambiado, existen barreras que impiden un cambio automático de los esquemas de trabajo. En los espacios industriales donde operan estos robots se dificulta la introducción de Internet, porque la mayoría de computadoras usadas para el control son obsoletas. Una migración del hardware de control no se justifica totalmente, porque no se agregaría mayor poder de producción, estos robots ya trabajan a su máxima capacidad. Sin embargo hay industrias que tienen estrategias de migración a Internet, pero no tienen la experiencia para dirigir proyectos de este tipo.

Necesidades actuales
En contraposición con las barreras para la migración, también se ha fortalecido la necesidad de incorporar nuevas funciones a los robots, tales como la capacidad para permitir el monitoreo y diagnóstico remoto, su integración a los procesos empresariales, retroalimentación del proceso de fabricación. El acceso a Internet, permite compartir estas herramientas entre las diferentes áreas de fabricación.

Planteamiento del problema

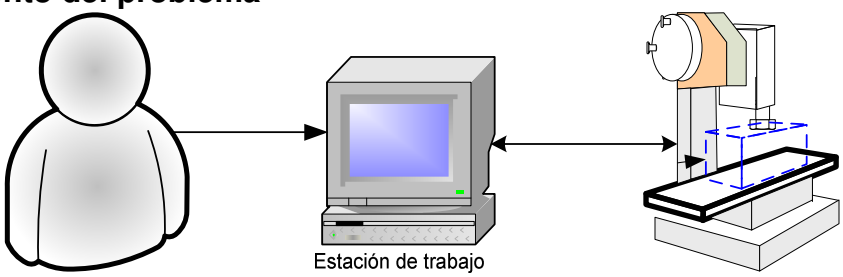


Figura 2 Operación local de robots CNC

No existe un protocolo estándar de comunicación con los robots CNC, porque se concibieron para ser operados localmente.

Los robots CNC se controlan y operan de forma local (Figura 2), en equipos de bajo poder de procesamiento. Se necesita permitir la operación de robots CNC de forma remota, mediante un esquema estandarizado, consistente y confiable, que pueda ser usado e implementado por desarrolladores de diferentes niveles de dominio de las tecnologías (Figura 3).

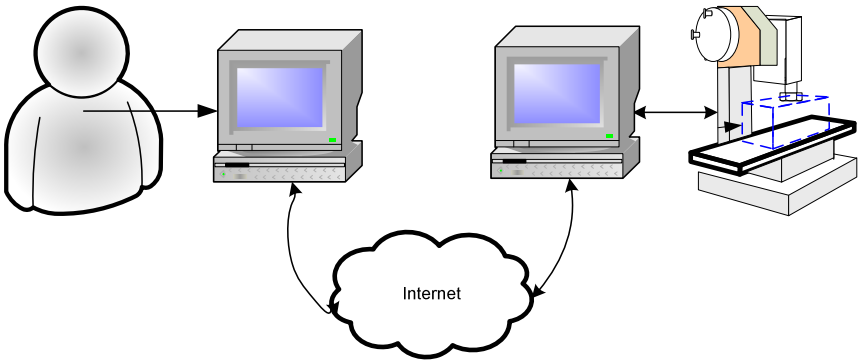


Figura 3 Teleoperación de robots CNC

Las soluciones comerciales presentan las desventajas propias de su licencia, el código fuente no está disponible para su adaptación a necesidades específicas, usan protocolos de comunicación propietarios, el mantenimiento y soporte técnico debe ser realizado por personal especializado.

El Software de control del robot MAXNC-10 opera de forma local, en la plataforma DOS, en Windows 95 o 98 se ejecuta en una ventana de MS-DOS.

1.4 Contribuciones

- Diseño de una infraestructura de bajo costo para la manipulación de robots CNC, basado en una arquitectura cliente-servidor, consumo bajo de recursos de los equipos, incluso puede ser ejecutado en equipos antiguos. (*Capítulo 4*).
- Arquitectura basada en patrones de diseño, que facilita la incorporación de nuevos tipos de controladores (*Sección 4.4.1*).
- Implementación de un protocolo de comunicación para la teleoperación de robots CNC vía Internet (*Sección 5.2.2.3*).
- Desarrollo de un producto de software para la teleoperación del robot MAXNC-10, que incluye los módulos para editar el código, simulación y control (*Secciones 5.2.3,5.2.4,5.2.5*).
- En el área de investigación, permitirá tener acceso al control de robots CNC vía Internet. El Framework puede aplicarse para implementar aplicaciones de manipulación de otras configuraciones de robots.

- El control puede aplicarse para manipular dispositivos de creación de circuitos electrónicos, tabletas XYZ y automatización de ensamblado, usando el lenguaje RS274-D.

1.5 Organización de la tesis

Capítulo 1. Define el objetivo general y los objetivos específicos que se deben alcanzar en el desarrollo de este trabajo. Presenta los antecedentes para el desarrollo de una arquitectura de teleoperación de robots CNC, se plantea el problema que se va resolver, lista las principales contribuciones como el alcance de los objetivos.

Capítulo 2. Se realiza una profunda investigación del estado del arte en la teleoperación de robots CNC, las diferentes alternativas de solución que se han propuesto para el problema expuesto en la sección 1.3, enfocándose en la teleoperación y control de robots CNC. Se realiza un análisis de las tecnologías candidatas para la implementación de la solución, identifica las fortalezas y debilidades de cada una y se realiza la selección más conveniente para alcanzar los objetivos expuestos en la sección 1.1.

Capítulo 3. Profundiza en el uso y aplicación de las tecnologías seleccionadas para la implementación de la solución, de tal forma que exista un marco conceptual que ayude a formular un análisis, diseño e implementación integral.

Capítulo 4. La construcción de la solución se apoya en el uso de las metodologías de ingeniería de software orientadas a objetos. Se realizan las actividades de análisis de los requerimientos (*4.3*) y el diseño de la arquitectura (*4.4.1*) para alcanzar el primer objetivo específico. Se realiza el análisis y diseño del módulo de control (*4.4.2.2, 4.5.1.2*) y el protocolo de comunicación (*4.4.2.3, 4.5.1.3*), definidos en los *objetivos específicos 2 y 3.* Mediante los casos de uso se especifican los alcances y funcionalidad que el sistema debe ofrecer a los diversos actores.

Capitulo 5. Se desarrollan las etapas de implementación y pruebas. Explica los algoritmos usados para la implementación del módulo de control (*5.2.2.2*), detalla el protocolo de comunicación para la teleoperación (*5.2.2.3*). Se atienden los *objetivos específicos 4 y 5*, mediante tres casos de estudio: Desarrollo de un driver para el robot MAXNC-10 (5.2.3), desarrollo de un driver para un robot virtual en OpenGL (5.2.4). El trabajo culmina integrando todos los elementos desarrollados a lo largo de la investigación en una aplicación de escritorio IDE-CNC (5.2.5). Las pruebas se realizan verificando el cumplimiento de los casos de uso del capítulo 4, por medio de la construcción de dos piezas mecánicas (5.3).

Capítulo 6. Se formulan las conclusiones a partir del conocimiento adquirido, dentro de las cuales se precisan los logros obtenidos en base a los objetivos planteados. Se identifican algunos puntos de referencia para continuar el trabajo en otros campos de aplicación.

Se incluye un glosario de términos y un breve listado de los acrónimos usados. Al final del documento se presenta una lista de las referencias consultadas, se incluye una sección con tres anexos que contienen la siguiente información: Anexo A) Gramática del lenguaje RS274-D usada para desarrollar el intérprete de comandos en la sección *5.2.2.2.1*, Anexo B) Diagramas de secuencia de los casos de uso descritos en la sección *4.4.2*, Anexo C) Secuencias de pasos en modo medio paso para el robot MAXNC-10 que se puede usar para modificar el driver desarrollado en la sección *5.2.3*.

Capítulo 2 Antecedentes y estado del arte en teleoperación de robots CNC

2.1 Introducción

En este capítulo se realiza un análisis de las tecnologías disponibles para la implementación de los objetivos específicos *2,3 y 5* de la tesis, dividido en tres grupos:

- Tecnologías de comunicación.
- Programación de gráficos 3D.
- Lenguajes de programación de robots CNC.

Se ha dejado para el final del capítulo el estado del arte en teleoperación de robots CNC.

2.2 Tecnologías de comunicación

Uno de los objetivos planteados en este trabajo es, *diseñar un protocolo de comunicación para la teleoperación de robots CNC*. Las soluciones de teleoperación se han implementado usando diversas tecnologías, en este proyecto se ha seleccionado la comunicación por medio de Internet por ser el protocolo de facto en las redes actuales.

Se pueden citar tres modelos de comunicación entre aplicaciones, el modelo cliente-servidor que se implementa normalmente por medio de sockets; llamadas a procedimientos remotos y el modelo basado en objetos distribuidos.

2.2.1 Sockets

La interfaz de comunicación por medio de sockets es un API para el desarrollo de aplicaciones sobre los protocolos TCP y UDP. Los sockets son puntos finales de enlaces de comunicación entre procesos. Para el sistema operativo Microsoft Windows® se desarrolló el API Winsock (Windows Sockets), soporta las rutinas básicas de la versión existente para Unix y se agregaron funciones para uso específico de esta plataforma.

Los sockets son ampliamente usados para implementar aplicaciones en el modelo cliente-servidor.

2.2.1.1 Arquitectura

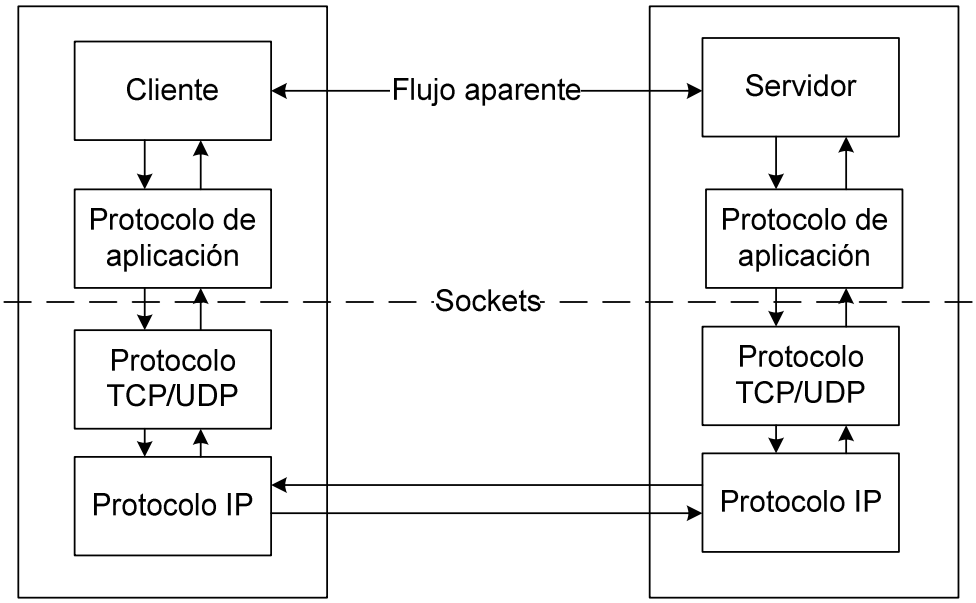


Figura 4 Arquitectura de sockets

Servidor. Abre un *socket* en un puerto bien conocido y espera peticiones de los clientes. Al recibir una instrucción ejecuta localmente el procedimiento asociado al comando y devuelve el resultado de acuerdo al protocolo de aplicación. El servicio puede ser una consulta a base de datos, enviar archivos, consultar un sistema o activar dispositivos.

Cliente. Cuando un cliente requiere un servicio, abre una conexión a la dirección IP y puerto del servidor. Envía las instrucciones definidas en el protocolo de aplicación, espera la respuesta y procesa los datos recibidos. Los resultados de la gestión de los datos enviados y recibidos, se entregan al usuario en el formato definido por la aplicación: pantalla, archivos o alguna otra forma de presentación.

Protocolo de aplicación. Es el conjunto de reglas sintácticas y semánticas que le dan sentido a la comunicación entre el cliente y servidor. Especifica cómo se

sincronizan los procesos. La aplicación tiene la libertad de construir su propio protocolo de acuerdo a sus propias necesidades.

Protocolo TCP/UDP. Es responsable del envío y recepción de los datos del protocolo de aplicación. El protocolo TCP ofrece un servicio orientado a conexión, en la que se mantiene la información de estado de cada conexión. El protocolo UDP ofrece un servicio orientado a la no conexión, la comunicación entre cliente y servidor no se mantiene activa, aunque puede ser más eficiente hay que tomar en cuenta que la entrega de datos no está garantizada.

Protocolo IP. Establece el sistema de direccionamiento lógico de red, se encarga de hacer llegar los paquetes a su destino.

2.2.1.2 Ventajas

- Sencillez de programación.
- Comunicación entre diferentes plataformas.
- Comunicación entre cliente-servidor desarrollados en diferentes lenguajes.
- No se necesita agregar ninguna capa de software.

2.2.1.3 Desventajas

- Se considera un mecanismo de bajo nivel.
- Se debe desarrollar un protocolo específico para la aplicación.

2.2.2 Llamadas a procedimientos remotos

Es una arquitectura que permite a un cliente invocar métodos implementados en servidores remotos, usando el mecanismo de llamadas a métodos locales [30]. El servidor responde al cliente RPC como si hubiera sido invocado localmente recibiendo la solicitud del cliente y atendiéndola sincrónicamente (Figura 5).

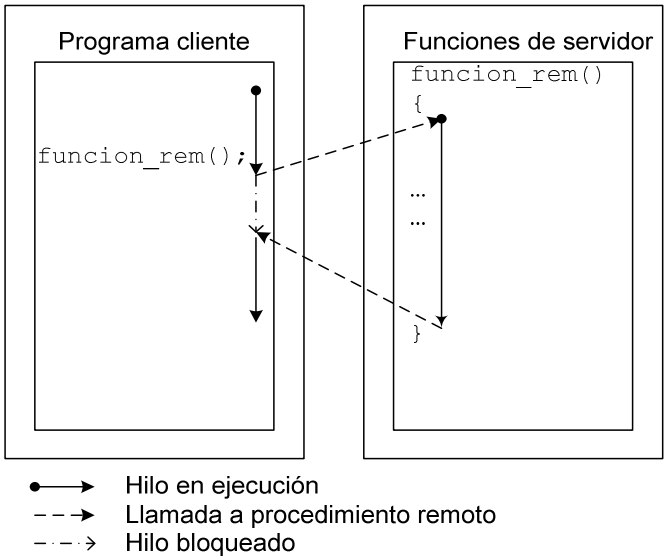


Figura 5 Hilo de ejecución de una llamada a un procedimiento remoto.

2.2.2.1 Arquitectura

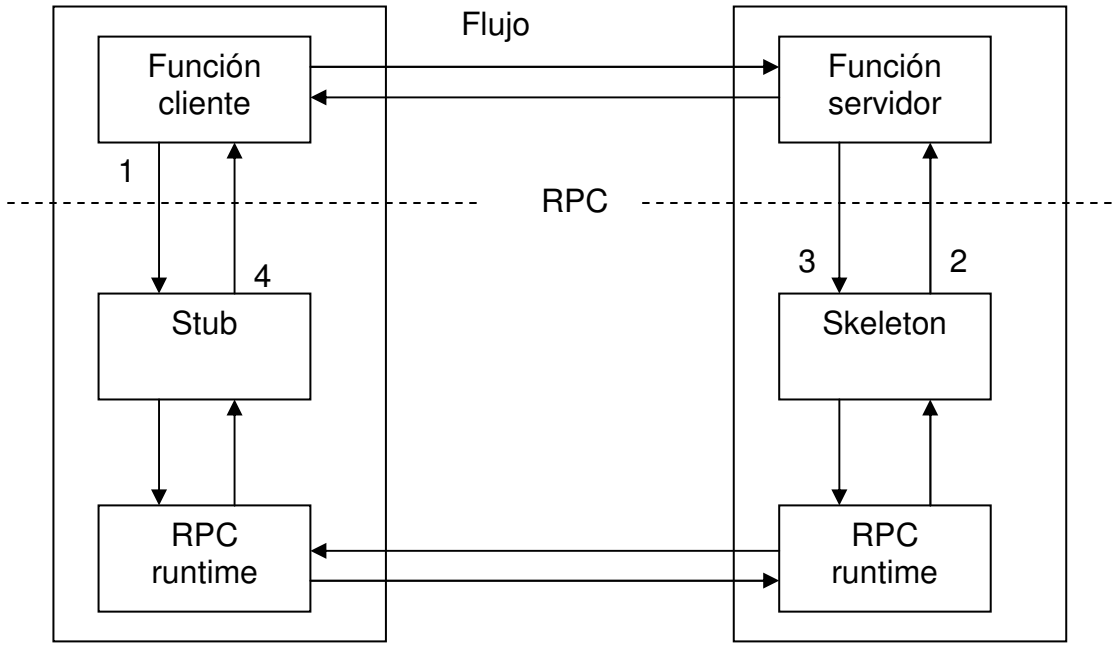


Figura 6 Arquitectura de llamadas a procedimientos remotos

Función cliente. Es la función local, se compila con un código adicional (stub), para crear una envoltura de llamada de sistema. Es la rutina que realizará la invocación de las funciones remotas.

Función servidor. Es la función remota, que se encargará de atender las solicitudes de invocación del cliente, también se compilan con un código adicional (skeleton).

Stub y skeleton. Son los subsistemas usados por los procesos cliente y servidor para transferir la solicitud a la capa de transporte. Realizan un proceso de empacado y organización de los argumentos para ser enviados a la capa subyacente.

RPC runtime. Es responsable de crear la independencia de protocolo.

2.2.2.2 Ventajas

- Crear una abstracción de comunicación basada en llamadas locales.
- No se necesita construir un protocolo de comunicación.

2.2.2.3 Desventajas

- No existe un estándar que defina claramente su funcionamiento[30].
- Los problemas de comunicación pueden confundir las causas reales de una falla[30].

- Programación complicada, cuando se manejan archivos o dispositivos que pueden llegar a fallar, en tal caso la programación debe ser planeada cuidadosamente.
- Cada cliente debe ser diseñado para interconectarse al servidor mediante el procedimiento apropiado.
- RMI es una tecnología para comunicar exclusivamente máquinas virtuales de Java.

2.2.3 Objetos distribuidos

En esta arquitectura se persigue eliminar la distinción entre cliente y servidor, los componentes fundamentales del sistema son objetos que proveen una interfaz a un conjunto de servicios. Los objetos se distribuyen a lo largo de varias computadoras y se comunican entre ellos a través de un *middleware*. Cada uno de los objetos está ubicado en su propio espacio de direcciones, pero aún así se comportan como si fueran uno solo.

2.2.3.1 CORBA (Common Object Request Broker Architecture)

Es una arquitectura abierta y diseñada en el paradigma orientado objetos. La comunicación entre los objetos distribuidos, se realiza mediante el protocolo IIOP. El *middleware* recibe el nombre de *Agente de Solicitud de Objetos* (ORB, Object Request Broker). El objetivo de esta arquitectura es facilitar la comunicación entre aplicaciones desarrolladas por diferentes fabricantes, en diferentes sistemas operativos, topologías de red y lenguajes de programación.

2.2.3.2 Arquitectura

Las aplicaciones CORBA se componen de objetos. Para cada tipo de objeto, se define una interfaz en lenguaje de definición de interfaces (IDL). La interfaz es el contrato que ofrece el servidor, cada cliente debe usar el mismo IDL para especificar las operaciones que desea invocar. El IDL es independiente del lenguaje de programación, existen versiones implementadas para los lenguajes C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python e IDLScript (Figura 7).

Gracias a esta separación entre interfaz e implementación, se logra la interoperabilidad, independencia del fabricante, sistema operativo o lenguaje.

2.2.3.3 Ventajas

- Uso de componentes desarrollados en otros lenguajes.
- Uso de objetos ejecutándose en diferentes plataformas.
- Compartir recursos de hardware o software para un atender una solicitud compleja.

2.2.3.4 Desventajas

- Incompatibilidad entre implementaciones.
- Es una arquitectura compleja de implementar.

- Se mezclan diversas tecnologías de varios vendedores.
- Dificultades para probar y depurar.

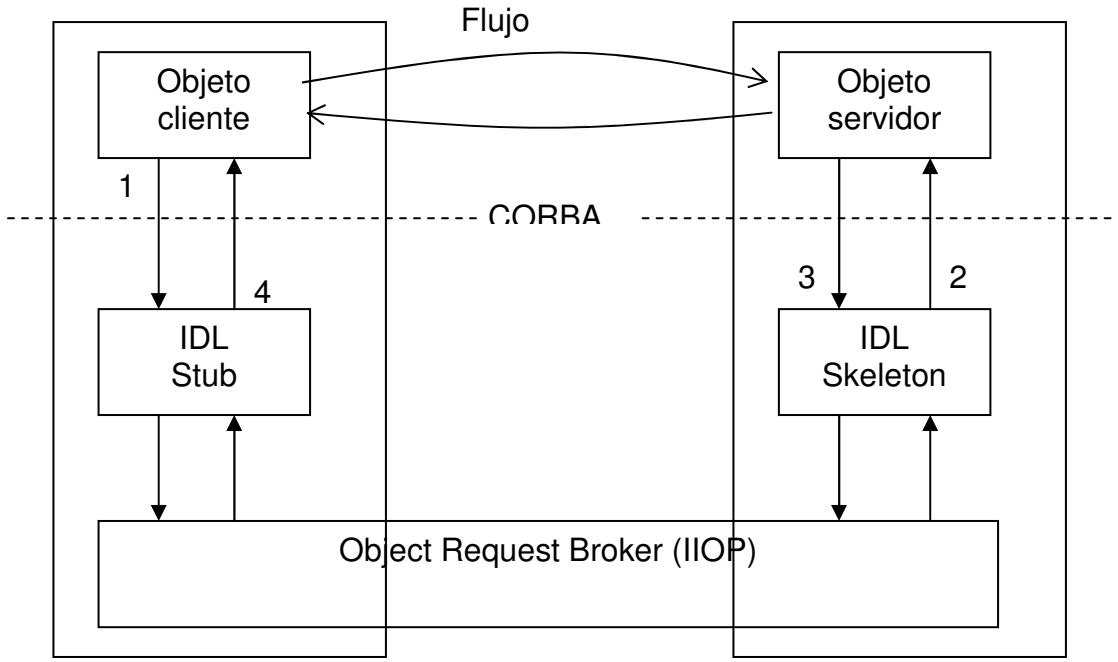


Figura 7 Arquitectura de CORBA

2.3 Programación de gráficos 3D

Otro de los objetivos específicos es, *diseñar una aplicación de escritorio que permita la edición, simulación y ejecución de comandos RS274-D*. Las tecnologías que se presentan a continuación han sido diseñadas específicamente para la presentación de gráficos en 3D, adecuadas para el desarrollo de sistemas de simulación.

OpenGL [35][13] y DirectX [14][20] son bibliotecas de funciones que facilitan el desarrollo de aplicaciones 3D. Adicionalmente se describe el paquete Java3D [25], que es la tecnología de Java para la programación en 3D, internamente usa las implementaciones de OpenGL y DirectX, en un paradigma llamado *scene graph*.

2.3.1 OpenGL

Es un API desarrollado por Silicon Graphics, está compuesto por una biblioteca de funciones para la construcción de objetos tridimensionales. La aplicación es controlada por una máquina de estados que se encarga de administrar una *tubería de presentación* tanto los atributos de la máquina de estados como las primitivas se envían a la máquina de estados por medio de las funciones. OpenGL requiere que el hardware de gráficos, tenga un *framebuffer, porque* a través de él se realizan las operaciones de dibujo. Los fabricantes de tarjetas de vídeo y

aceleradoras gráficas, implementan en hardware las funciones definidas en el estándar OpenGL.

2.3.1.1 Modo de operación

La Figura 8 muestra un diagrama esquemático del modo de operación de OpenGL. Los comandos ingresan en el lado izquierdo, se pueden especificar objetos geométricos u objetos que controlarán el comportamiento de la representación de los objetos geométricos.

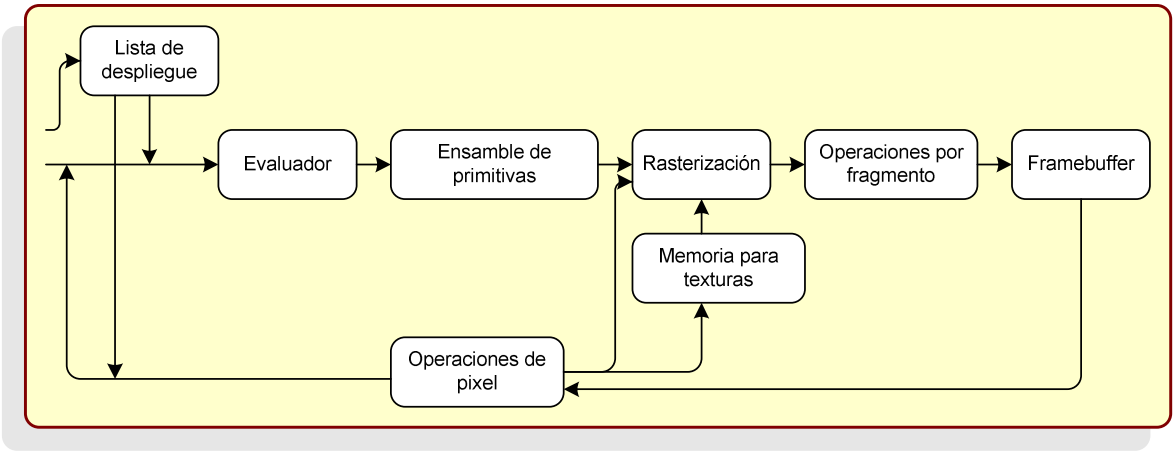


Figura 8 Modo de operación de OpenGL

Tabla 1 Descripción de los componentes del modo de operación de OpenGL

Componente	Descripción
Lista de despliegue	Los datos que describen píxeles o geometrías, se guardan en lista de despliegue para ser usada inmediata o posteriormente.
Evaluador	Los evaluadores proporcionan un método para obtener los vértices que representan curvas paramétricas y superficies definidas por puntos de control y funciones polinomiales.
Ensamble de primitivas	Para datos representados con vértices, los convierte a su representación en primitivas. Los datos espaciales son convertidos en matrices de punto flotante de 4x4.
Operaciones de píxel	Los píxeles que se obtienen de un arreglo en memoria son desempacados y se realizan operaciones de transformación. A continuación se envían a la memoria de texturas o enviados al estado de rasterización. Si los píxeles se obtienen del framebuffer, se realizan operaciones de transformación.

Componente	Descripción
Rasterización	Es la conversión de datos geométricos y píxeles en fragmentos. Cada fragmento cuadrado corresponde a un píxel en el framebuffer.
Memoria para texturas	Es el espacio de memoria donde se pueden cargar los objetos de texturas, que se aplicarán a los objetos geométricos. Esta memoria facilita y acelera su intercambio.
Operaciones por fragmento.	Antes de enviar los datos al framebuffer, se realizan una serie de operaciones que pueden alterar o ignorar fragmentos. Las operaciones realizadas en secuencia son: texturizado, difusión, suavizado, sombreado, operaciones lógicas y enmascaramiento de bits. Al final de las operaciones se envía el fragmento procesado al buffer apropiado.
Framebuffer	Es el conjunto de buffers de un sistema. Cada buffer es usado para almacenar datos de un mismo tipo correspondientes a un píxel. Un buffer que almacena solo un bit de información se llama bitplane. El buffer de color es el único que se puede acceder vía programación.

Ventajas

- Disponible en plataformas Windows, Unix, Linux, Mac y otras.
- Aprovecha las funciones de aceleración implementadas en hardware cuando están disponibles.
- Es un estándar ampliamente usado.

Desventajas

- Es un API exclusivamente para presentación de gráficos en 3D, no incluye el soporte de otros dispositivos, audio, video, streaming media.
- No es orientado a objetos.

2.3.2 DirectX

Es un conjunto de tecnologías para el desarrollo de aplicaciones multimedia desarrollada por Microsoft, para las plataformas Windows 98 y superiores. El API se define por medio de interfaces COM.

La versión más reciente es DirectX9.0. Microsoft Direct3D es la tecnología de gráficos 3D. En esta versión surge DirectX Graphics que es en realidad la integración de dos tecnologías Directdraw y Direct3D que estaban separadas en versiones anteriores.

2.3.2.1 Modo de operación

El proceso de creación de un escenario con Direct3D se muestra en la Figura 9. Los comandos ingresan por el lado izquierdo y siguen conceptualmente la secuencia indicada en el diagrama.

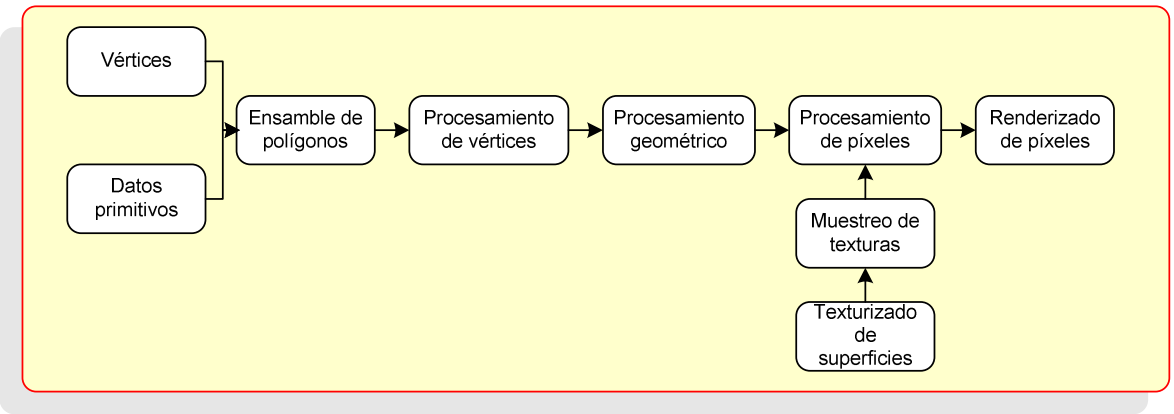


Figura 9 Modo de operación de Direct3D

Tabla 2 Descripción de los componentes del modo de operación de Direct3D

Componente	Descripción
Vértices	Modelos definidos por vértices sin transformaciones. Se almacenan en los buffers de vértices.
Datos primitivos	Es una colección de vértices que forman una entidad 3D simple. Puntos, líneas, triángulos y polígonos.
Ensamble de polígonos	Convierte primitivas de alto nivel, mapas de desplazamiento y lados de polígonos a localidades de vértices, almacena estas localidades en los buffers de vértices.
Procesamiento de vértices	Conjunto de transformaciones, que se aplican a los vértices almacenadas en el buffer de vértices.
Procesamiento geométrico	Operaciones de corte, mascara, evaluación de atributos y rasterización, aplicada a los vértices transformados.
Texturizado de superficies	Aplicación de una textura a un objeto. Las texturas son imágenes planas, por lo tanto deben ser ajustadas a la superficie del objeto.
Muestreo de texturas	Se aplica un filtro de nivel de detalle de textura, al valor de entrada de la textura.
Procesamiento de píxeles	Conjunto de operaciones aplicadas a los datos geométricos para modificar el vértice y la textura, para obtener los valores de salida del píxel.
Renderizado de pixeles.	Último paso del proceso de *renderizado*, modifica los valores del píxel con parámetros de tonalidad, profundidad, tipo de pluma o aplicación de efectos. Los valores obtenidos se envían a la pantalla.

2.3.2.2 Ventajas

- Aprovecha las funciones de hardware.
- Forma parte de DirectX, que es un conjunto de bibliotecas de funciones para el desarrollo de aplicaciones multimedia con soporte de dispositivos de entrada, audio, trabajo en red.

2.3.2.3 Desventajas

- Dependiente de la plataforma Windows.
- No es un estándar ampliamente usado.
- Es un API demasiado compleja.

2.3.3 Java3D

Java 3D es la tecnología de Java para la programación de gráficos 3D. Está desarrollada usando JNI para acceder a las implementaciones nativas de los API's OpenGL o DirectX. La biblioteca Java3D es dependiente de la plataforma, pero existen implementaciones específicas para las plataformas IBM AIX, AIX, HP-UX, Linux, SGI y MAC. La implementación de Java3D sobre OpenGL es más estable que la versión basada en DirectX.

2.3.3.1 Modo de operación

Se basa en un concepto de escenario gráfico compuesto de objetos individuales llamados Nodos, el escenario gráfico se construye a partir de un *universo virtual*, este tiene asociado un objeto *Locale* que define la precisión de las coordenadas y un conjunto de ramas. Cada rama es un sub-escenario gráfico que tiene como raíz un objeto Brachgroup. Cada objeto Brachgroup tiene un conjunto de hojas que implementan los objetos geométricos (Figura 10).

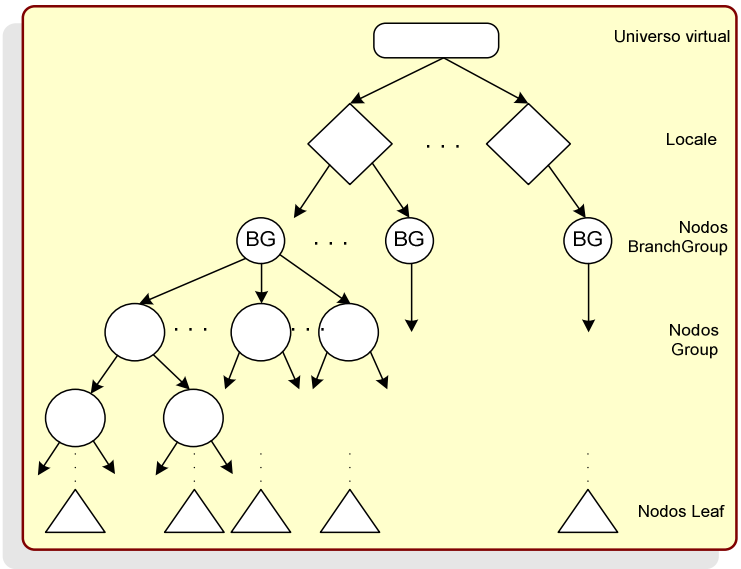


Figura 10 Modo de operación de Java3D

Tabla 3 Descripción de los componentes del modo de operación de Java3D

Componente	Descripción
Universo virtual	Es un espacio tridimensional que tiene asociado un conjunto de objetos. Los objetos solo pueden estar asociados a un universo virtual a la vez.
Locale	Actúa como un contenedor para una colección de subgráficos, especifica las coordenadas en alta resolución del escenario. Está compuesta por las coordenadas de un punto y un conjunto de ramas (Objetos BranchGroup).
Branchgroup	Es la raíz de un subgráfico de la escena. Puede ser compilado como una unidad, ligado al universo virtual o ser incluido como nodo hijo de un grupo en otro subgráfico. Cada objeto Branchgroup puede estar ligado a un solo nodo padre.
Group	Es un objeto de propósito general, que sirve para la agrupación de otros objetos. Los nodos Group, tienen exactamente un nodo padre y pueden tener 0 o más hijos.
Leaf	Contienen la definición de los objetos geométricos, luces, difusión, sonido y cualquier otra información que no pueda agrupar otros objetos. Los nodos Leaf, no tienen hijos y están ligados a un solo nodo padre.

2.3.3.2 Ventajas

- Orientado a objetos, lo que permite concentrarse en el modelo más que en la implementación.
- Soporte para multiproceso, internamente es multihilos, para aprovechar mejor los entornos multiprocesador.
- Soporte para dispositivos de realidad virtual como cascos y lentes.
- Al ser una capa externa sobre OpenGL o Direct3D, el programador se despreocupa de las diferencias entre ambas tecnologías.
- El código es 100% portable entre la implementación de OpenGl y DirectX.
- Es un API de más alto nivel que OpenGL o Direct3D.
- Acceso a toda la tecnología Java, soporte para audio, video y difusión de medios.
- Incluye soporte para leer archivos en diferentes formatos.

2.3.3.3 Desventajas

- El rendimiento de una aplicación Java3D es hasta 2.5 veces más lento que uno desarrollado en OpenGL, a pesar de que el procesamiento de las rutinas se ejecutan en su mayoría vía hardware.
- Las características gráficas que soporta son un subconjunto de las que ofrecen OpenGL y DirectX, solamente las que son comunes a ambas.
- Java3D es totalmente dependiente de las versiones de OpenGL o DirectX, si tienen algún defecto no se pueden corregir desde java.

- A pesar de que una aplicación 100% Java es totalmente portable entre las plataformas con máquinas virtuales Java, bibliotecas de funciones como Java3D, no lo son, ya que ellas usan funciones nativas.
- Oculta el mecanismo de rendering, lo que impide el acceso directo al framebuffer, haciéndolo inoperante para algunos proyectos.

2.4 Lenguajes de control CNC

El lenguaje de control numérico, es la base para permitirle al usuario programar las instrucciones al robot, es independiente del hardware. Se evalúan dos alternativas adicionales al lenguaje RS274-D.

2.4.1 HPGL Hewlett-Packard Graphics Language

Es un lenguaje de gráficos 2D, diseñado por Hewlett-Packard, específicamente para el control de plotters de plumillas, es un subconjunto del lenguaje HP PCL Nivel 5 [10].

Es un lenguaje sencillo, claro y fácil de leer. Es ampliamente usado para el intercambio de datos entre aplicaciones.

2.4.1.1 Descripción

Los instrucciones HPGL se especifican básicamente por medio de comandos individuales de dos caracteres ASCII, seguidos de cero o más parámetros y un carácter terminador normalmente punto y coma (;). Todos los comandos HPGL tienen la misma sintaxis.
Ejemplo:

```
PA 2000, 3000;
```

Esta instrucción mueve la pluma del plotter a la posición absoluta X=2000 y Y=3000.

Aunque este es un lenguaje muy estable y libre de errores conocidos, ya son muy pocos los fabricantes de plotters de plumillas[10], no maneja instrucciones para las tres coordenadas X,Y,Z.

2.4.2 RS274-X Formato Gerber Extendido

Es el lenguaje estándar para el diseño de circuitos electrónicos. La mayoría de photoploters tienen la capacidad de interpretarlo. Es una extensión del formato RS274-D, agrega funciones para el relleno de polígonos, composición de imágenes en positivo-negativo, apertura personalizada. Permite incluir parámetros de configuración en el encabezado del archivo.

2.4.2.1 Descripción

Es una estructura que consiste en coordenadas X, Y indicadas por comandos que definen donde empieza la imagen, la forma y donde termina. Además indica información de apertura, define la forma y tamaño de las líneas y orificios.

El archivo de comandos contiene parámetros RS274X y códigos estándar RS274-D. El archivo está compuesto por *bloques de datos* que contienen parámetros y códigos. Cada bloque de datos está acotado por un delimitador, comúnmente un asterisco (*), los bloques de códigos se escriben en una sola línea. Los bloques de datos pueden ser agrupados en capas.

Ejemplo:

G01X1Y1D02*

Esta instrucción mueve la herramienta del photoplotter linealmente (G01), inicia en la posición actual y termina en la coordenada X=1, Y=1, con el haz de luz de la herramienta en modo apagado (D02).

2.4.3 RS274-D G-Code

Es el estándar recomendado para programar máquinas CNC. Proporciona un conjunto de comandos para escribir programas de control numérico, seleccionar herramientas, modificar la velocidad y tipo de corte.

2.4.3.1 Descripción

La unidad básica de programación es el *bloque* de información. Cada bloque de información puede iniciar con un número de línea, seguido de un comando y opcionalmente algunos parámetros. Un comando se define por una letra y un par de dígitos. Los parámetros se definen con una letra y un número real. Los espacios en blancos son ignorados.

Ejemplo:

N001 G01 X0.0001 Y1.000 Z0.001

La instrucción del bloque N001, mueve la herramienta del robot CNC linealmente (G01), inicia en la posición actual y termina en la coordenada X=0.0001, Y=1.000, Z=0.001.

2.5 Estado del arte

El primer dispositivo que permitió realizar una tarea a distancia, fué desarrollado por Goertz [7] para manipular material radioactivo. Posteriormente se realiza el primer sistema para manipular un robot industrial (ABB 1400 de 6 grados de libertad) desde Internet, en la University of Western Australia [51], este sistema continúa en operación desde 1994.

A continuación se presentan los trabajos más importantes relacionados con la teleoperación de robots CNC. Se han clasificado de acuerdo a sus características, aunque algunos podrían estar en más de un grupo.

2.5.1 Soluciones de arquitectura de abierta

Proporcionan una alternativa para la integración de tecnologías heterogéneas, permitiendo la interacción, interoperabilidad, portabilidad y escalabilidad. La arquitectura para *el control de sistemas en tiempo real* (RCS, Albus et al, 1979) desarrollada en el NIST, marca el inicio de los modelos de control de arquitectura abierta.

2.5.1.1 OSACA

El objetivo de OSACA (System Architecture for Controls within Automation Systems) es la especificación de una *arquitectura abierta* para sistemas de control, que incluya funciones de control numérico para robots (CNC), dispositivos de control lógico programable (PLC) y control de celdas (CC). Que sirva de base para todos los sistemas de automatización, que sea escalable a nuevas funciones y configuraciones de cómputo [44].

El proyecto OSACA es desarrollado y mantenido por The Osaca Project Consortium, que esta formada por universidades, fabricantes y vendedores Europeos.

2.5.1.1.1 Arquitectura.

La plataforma está compuesta por el hardware (Componentes electrónicos). El software incluye: el sistema operativo, sistemas de comunicación, drivers y opcionalmente programas del sistema. Los servicios del sistema pueden ser utilizados únicamente por medio del API, este oculta la implementación de los servicios de la plataforma, lo que permite que sea independiente de hardware o sistema operativo (Figura 11).

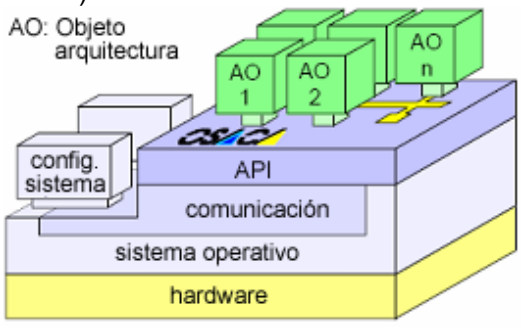


Figura 11 Arquitectura OSACA

2.5.1.2 OMAC

OMAC (Open Modulate Architecture Controllers) es una arquitectura que define las pautas para la creación de controladores desarrollados por diferentes

fabricantes, los cuales deben construir módulos compatibles y en forma de componentes. Esta arquitectura debe cumplir los requerimientos de ser abierta, modular, extensible, portable, escalable, mantenible y económica [40].

Esta arquitectura surge en 1994, con la publicación "Requirements of Open, Modulate Architecture Controllers for Applications in the Automotive Industry" [41], por parte de las empresas Chrysler, Ford and General Motors. Actualmente es definida por el consorcio TEAM (Technologies Enabling Agile Manufacturing), una organización con participación del gobierno, industria y academia de los Estados Unidos.

2.5.1.2.1 Arquitectura

Está integrada modularmente por los siguientes componentes: Interprete de programas, coordinador de tareas, grupo de ejes, coordinador de trayectorias, modelos de cinemática, agentes de interfaz de usuario, puertos de I/O. En la Figura 12 se muestra un ejemplo de implementación de esta arquitectura.

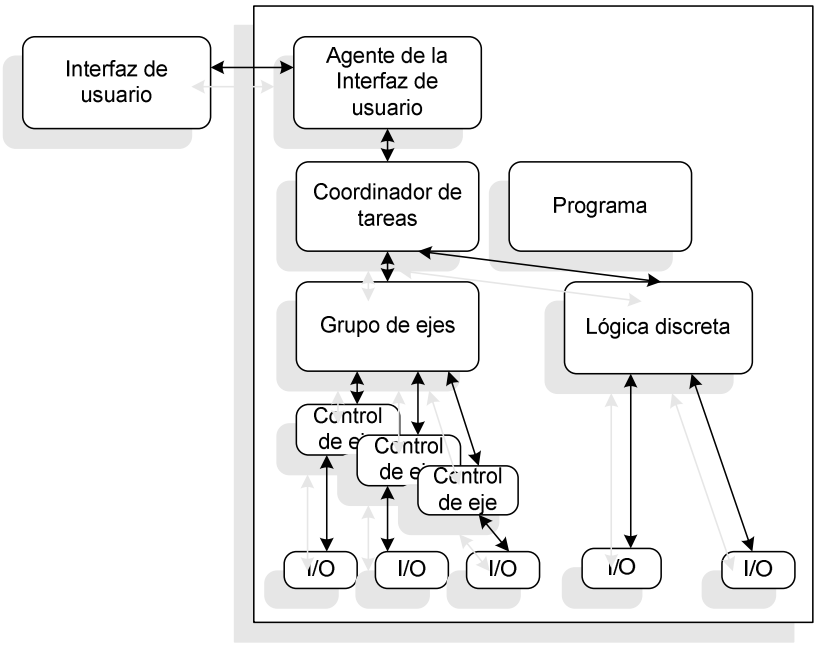


Figura 12 Arquitectura OMAC

2.5.1.3 OSEC

OSEC (Open System Environment for Controller) [43], es una arquitectura abierta, diseñada para trabajar en una computadora personal y el sistema operativo Windows para mejorar el rendimiento y facilitar el mantenimiento. La computadora controla el robot y además puede actuar como una fuente de información para las operaciones de fabricación. Los elementos de esta arquitectura sirven también como elementos de soporte logístico. Es una arquitectura diseñada específicamente para equipo de control numérico.

Esta arquitectura es el resultado del trabajo de un grupo de seis compañías japonesas: Toshiba Machine Co., Toyoda Machine Works Ltda., Yamazaki Mazak Co., IBM Japan Ltda., Mitsubishi Electric Co., SML Corporation.

2.5.1.3.1 Arquitectura

El modelo OSEC define 7 capas de referencia. (Máquina, Actuador de Servomecanismos, Movimiento, Control de procesos de la máquina, Planeación de la operación, Capa de diseño). Describe la relación con las actividades CAD/CAM. Se construyeron servicios y protocolos para distribución de sus componentes. Se agregó el lenguaje de descripción de equipos de fabricación automática (FADL, Factory Automation Equipment Description Language) para mejorar la interfaz entre los sistemas CAD/CAM y los equipos CNC.

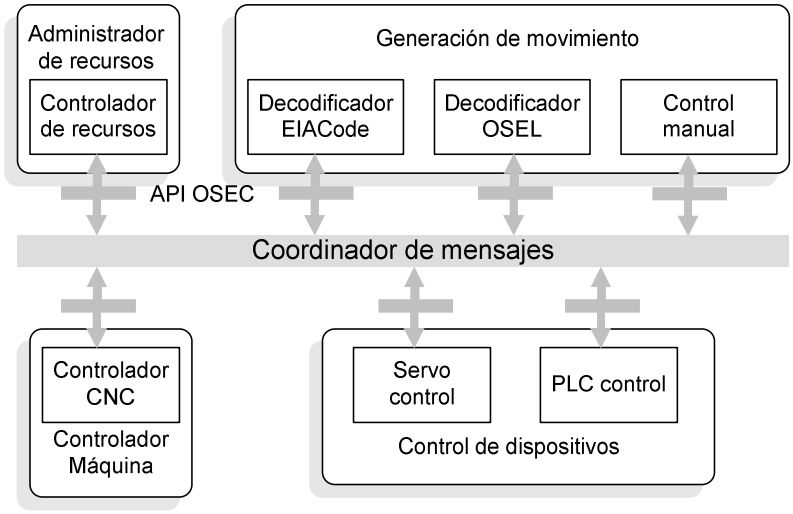


Figura 13 Arquitectura OSEC

2.5.1.4 EMC

El NIST, en colaboración con un conjunto de industrias y centros de investigación, trabaja en la investigación de la aplicación de los conceptos de arquitecturas abiertas a los controladores de maquinas CNC.

EMC (Enhanced Machine Control) [47], es un software para el control de robots cartesianos y no cartesianos, como robots hexápodos. El control de movimientos puede operar servomotores analógicos con una estrategia de lazo cerrado implementado en el software EMC o lazo abierto para motores a pasos o servomotores de paso. El software se basa en la metodología RCS y se programó usando las bibliotecas NIST RCS, corre en el sistema operativo de tiempo real QNX.

Este software fué diseñado originalmente por la división de sistemas inteligentes del NIST en los Estados Unidos, actualmente se encuentra disponible como un proyecto de código abierto. Las arquitecturas ISAM y MSI, también fueron

diseñadas por el NIST, basadas en el modelo RCS, con aplicaciones diferentes pero conceptualmente similares.

2.5.1.4.1 Arquitectura

Los cuatro componentes más importantes de una implementación de EMC son: Un controlador de movimiento (EMCMOT), un controlador discreto de I/O (EMCIO), el ejecutor de las tareas (EMCTASK) y una colección de interfaces de usuario en modo texto o gráfico.

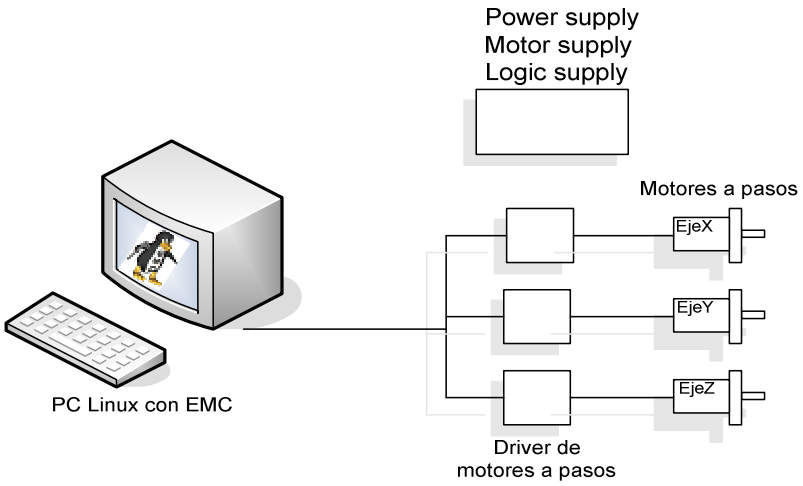


Figura 14 Robot CNC controlado por EMC

La Figura 14 muestra un diagrama de bloques de un sistema sencillo controlado por EMC, se puede ver un sistema de motores a pasos, la computadora con Linux en tiempo real y EMC, el envío de señales a los motores se realiza por el puerto paralelo.

2.5.2 Solución RPC

2.5.2.1 TACO

TACO (Telescope and Accelerator Control Objects) es un sistema de control de dispositivos desarrollado por el ESRT (European Synchrotron Radiation Facility) en el paradigma orientado a objetos. El modelo trata a todos los componentes a controlar como objetos que pueden ejecutar comandos. Los objetos se llaman *dispositivos* y se ponen a disposición a lo largo de la red. Los dispositivos se crean y almacenan en servidores de dispositivos. Los comandos que pueden ser ejecutados se implementan en una clase derivada de la clase *device*. Los comandos son invocados a través de un API en C++.

Arquitectura.

El diseño de TACO [2] se basa en el modelo de llamadas a procedimientos remotos. Todos los *dispositivos* son creados y ofrecidos por los *servidores de dispositivos*. Los clientes acceden a los dispositivos por medio de un API de programación de red (DSAPI, Device Server Application Programmer's Interface).

El control de todos los servidores y clientes se realiza en el Servidor de administración. La configuración se almacena en el Servidor de base de datos (Figura 15).

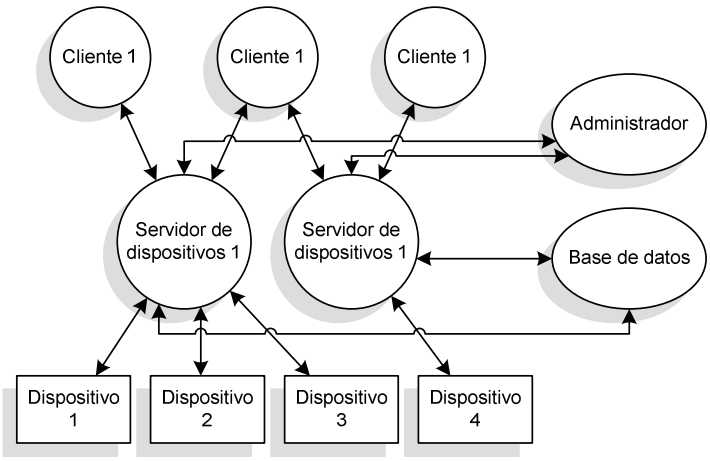


Figura 15 Arquitectura básica de TACO

Todas las llamadas de red entre clientes y servidores de dispositivos se implementan usando ONC/RPC (Sun Open Network Computing / Remote Procedure Call).

2.5.3 Soluciones CORBA

2.5.3.1 Control de máquinas basado en CORBA

El OMG ha propuesto un estándar para el control de robots de manufactura, desde el contexto del proceso de fabricación y su relación con los demás entornos de la organización, para lo cual propone las interfaces que permitan su integración. El OMG no realiza implementaciones en software, solamente define los estándares de trabajo.

2.5.3.1.1 Modelo conceptual

Desde el punto de vista del OMG el control de robots es un entorno combinado de hardware y software, responsable de controlar las operaciones de fabricación. El siguiente modelo muestra las principales entidades participantes en el proceso de fabricación (Figura 16).

Las entidades que están involucradas activamente con el control de los robots son: proceso de operación, robot y los controladores.

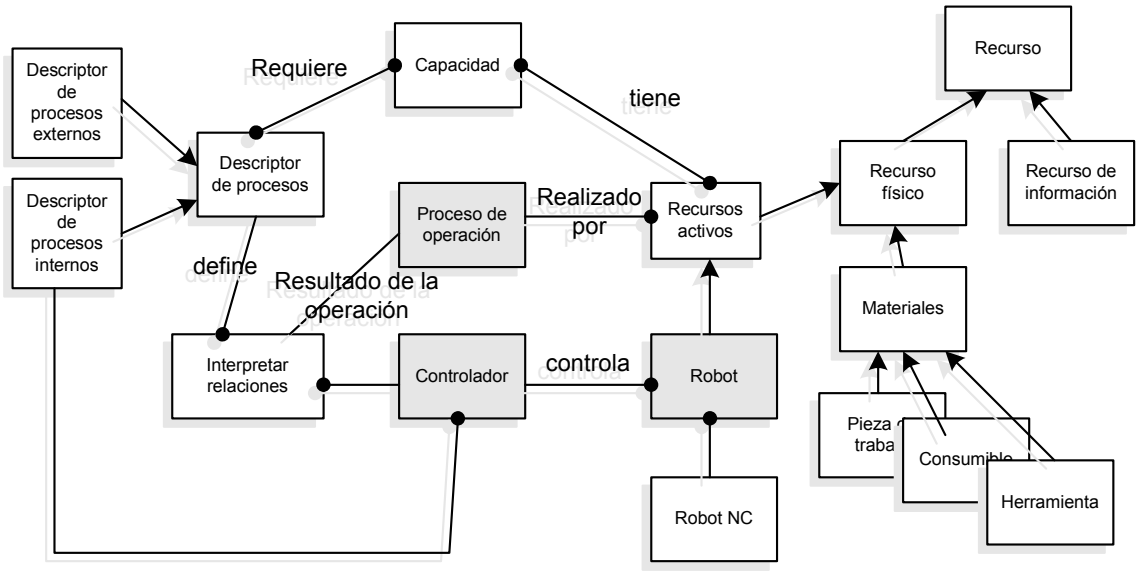


Figura 16 Modelo conceptual del control basado en CORBA

2.5.3.2 SAMT

El SAMT (Sandia Agile Manufacturing Testbed) [8] del Sandia National Laboratories, ha diseñado una arquitectura de fabricación basado en CORBA. La arquitectura define un objeto de software por cada objeto físico (torno, fresadora, taladro, etc.) mediante una interfaz IDevice. Los objetos CORBA permiten encapsular las características de un dispositivo, el controlador y la interfaz de red.

2.5.3.2.1 Modelo conceptual.

El objetivo de este proyecto es desarrollar un proceso de fabricación dinámica para diferentes maquinas y productos. El ciclo de fabricación (Figura 17), incluye el diseño, planeación, controlador de la célula y fabricación.

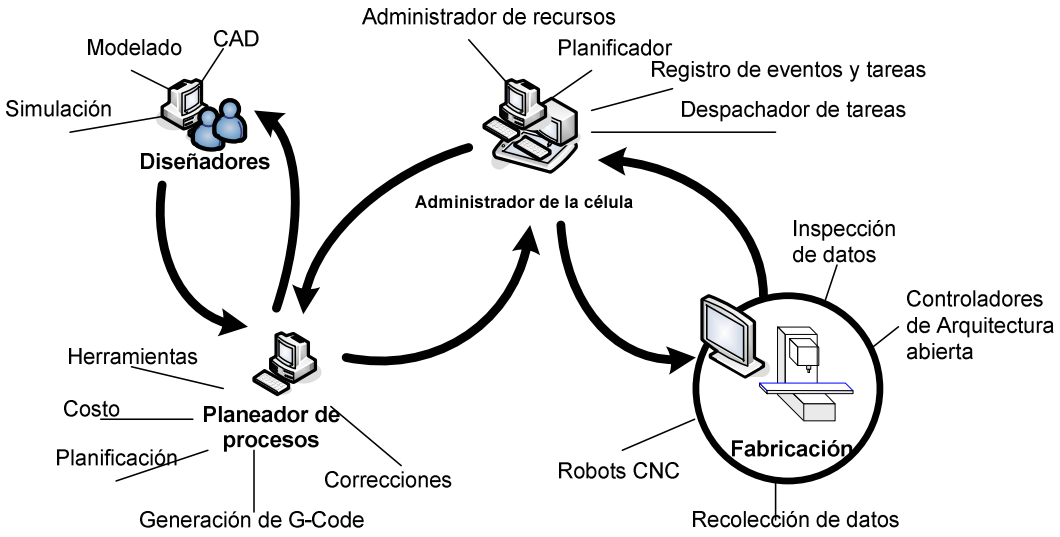


Figura 17 Proceso de realización de un producto en SAMT

2.5.3.3 TANGO

TANGO **(**TAco Next Generation Objects**)**[1] es un sistema de control basado en CORBA desarrollado por el European Synchrotron Radiation Facility(ESRF). El modelo de objetos en TANGO soporta métodos, atributos y propiedades, todos los objetos son representaciones de dispositivos. Los dispositivos pueden estar en la misma computadora o distribuidos en un conjunto de computadoras conectadas en red. La comunicación en la red se realiza por medio de CORBA. La programación de los objetos se puede realizar en C++, Java y Phytón. TANGO Es la evolución de TACO, una arquitectura desarrollada con RPC's.

2.5.3.3.1 Características.

- **Archivo IDL.** Un solo archivo IDL que define las interfaces de red.
- **Patrón de dispositivos.** Es una estructura de clases que se usan para implementar el control de los dispositivos.
- **Multithreading.** Soporta multihilos a nivel de ORB y cliente-servidor.
- **Atributos+Propiedades.** Adicionalmente a los dispositivos, TANGO permite manejar datos normalizados.
- **Base de datos.** Usa la base de datos MySQL, para el manejo de datos persistente.
- **Naming.** Es un servicio proporcionado por el ORB, para la localización y nombrado de objetos.
- **Tipos de datos.** Se usan los proporcionados por CORBA, para transferir datos entre comandos y atributos.
- **Monitores.** Sirven para informan al usuario de los eventos generados por los dispositivos.
- **Device caching.** Es un mecanismo que acelera la lectura de dispositivos cuando estos son demasiados.
- **API.** Biblioteca para implementar las funciones propias de tango, está disponible en Java y C++.

2.5.4 Soluciones comerciales

2.5.4.1 OpenCNC®

OpenCNC es un software de arquitectura abierta [42] desarrollado sobre la plataforma y tecnologías Microsoft para controlar robots CNC. Es independiente del fabricante y de las tarjetas de control, realiza la implementación de todas las operaciones de control en el software. Precio aproximado $13,000.00 Dlls.

2.5.4.1.1 Arquitectura

Está compuesta por el software de control de robots CNC, componente para la operación distribuida de robots (Figura 18) y un API para el desarrollo de aplicaciones personalizadas.

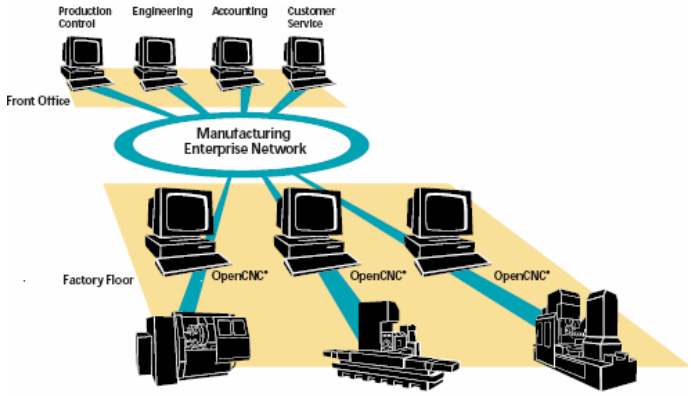


Figura 18 Operación distribuida de robots CNC con OpenCNC

2.5.4.2 SuperCam®

Es un sistema desarrollado para operar en modo DOS o Windows 98® desde una ventana DOS, controla los puertos paralelos directamente y permite manipular diferentes marcas de máquinas CNC, incluye opciones de dibujo y permite importar archivos Autocad. No tiene la opción de conectarse por medio de red. Precio aproximado $495.00 Dlls.

2.5.4.3 Xpert DNC®

Es una herramienta de software diseñado específicamente para permitir la comunicación de los robots CNC ubicados en el área de fabricación. Ofrece varios mecanismos de comunicación: por medio del puerto serial, mediante un modelo cliente-servidor sobre TCP/IP y por medio de una red ethernet usando puertos virtuales seriales.

2.5.4.4 CamSoft - CNC Profesional®

Es un producto desarrollado para operar en plataformas Windows, incluye software para el mantenimiento, diagnostico remoto monitoreo vía el puerto serial, componentes ActiveX o tarjeta de red. También permite recolectar información para los procesos de manufactura como tiempo de procesamiento, velocidad y tiempos de configuración. La versión estándar de este software tiene un costo de $4,999.00 Dlls.

2.5.5 Otras soluciones

Para el entrenamiento en el uso de las máquinas de control numérico se ha desarrollado un Sistema de entrenamiento para operación de máquinas CNC, basada en realidad virtual [22]. El modelo está compuesto por tres elementos principales: Máquina CNC, panel de control y pieza de trabajo.

2.5.5.1 Paradigma de operación de robots basado en Internet[54].

Es una filosofía de trabajo que se aplica en la teleoperación de robots, estos son considerados estaciones de Internet con capacidad de movimiento. Diferentes usuarios con distintos roles se pueden conectar al sistema simultáneamente. Este

es un proyecto en nacimiento, aún se está seleccionando la tecnología de implementación.

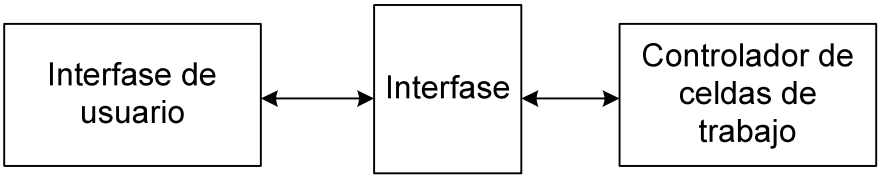


Figura 19 Operación de robots basado en Internet

En este paradigma de trabajo (Figura 19) se asume que existe un canal de comunicación basado en Internet, y que los tres componentes del modelo pueden interactuar a través de él. El controlador de celdas de trabajo, se puede entender como un servidor de Internet, con la capacidad de recibir movimientos en un espacio de coordenadas y hacerlas llegar a un manipulador o cualquier otro dispositivo.

Resumen

Las tecnologías seleccionadas para un cumplimiento óptimo de los objetivos de la tesis son las siguientes: Lenguaje **RS274-D** para el control de robots CNC; para la construcción del protocolo de comunicación, los **Sockets** ofrecen un entorno sencillo, flexible y poderoso; **OpenGL** es el ambiente adecuado para el desarrollo de aplicaciones gráficas en 3D, por ser multiplataforma y el estándar en las tarjetas gráficas.

En el estado del arte en teleoperación de robots CNC, se han analizado las diferentes estrategias para abordar el problema de teleoperación de los robots CNC. Se usan tres mecanismos de comunicación principalmente: Cliente servidor, llamadas a procedimientos remotos y objetos distribuidos.

En el siguiente capítulo se abordan a detalle cada una las tecnologías seleccionadas, a fin de contar con los elementos necesarios para proponer un modelo sólido.

Capítulo 3 Marco conceptual de las tecnologías para teleoperación de robots CNC

3.1 Introducción

En este capítulo se describen las tecnologías seleccionadas que se usarán en la etapa de implementación de los objetivos específicos 2,3, 4 y 5.

3.2 Programación con sockets

En el proceso de comunicación en red, participan en los *extremos* dos computadoras o procesos que se envían datos entre si. El socket es una representación abstracta (Figura 20) de los extremos en el proceso de comunicación [30][33].

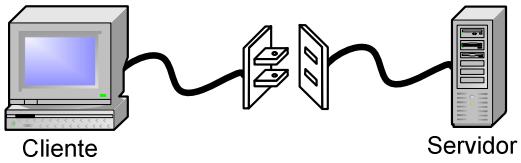


Figura 20 Representación abstracta de los sockets

La interfaz de sockets, es un API de programación que define las funciones para establecer la comunicación con un equipo remoto en un determinado puerto, atender solicitudes de conexión y enviar, recibir y publicar información.

3.2.1 Descripción del proceso

Una conexión de red por medio de sockets requiere de tres elementos de información; un protocolo, una dirección IP y puerto local, una dirección IP y puerto

remoto. El socket debe ser configurado para que desempeñe funciones de cliente o servidor.

3.2.1.1 Tipo de comunicación

La conexión entre dos sockets puede ser orientada a conexión o sin conexión. Para la comunicación orientada a conexión se usa el protocolo TCP. Los datos viajan como un flujo de bytes (*streams*), el protocolo TCP garantiza que los datos llegan completos y en el orden en que fueron trasmitidos. Una vez que los dos sockets estén conectados, se puede utilizar el canal para transmitir datos en ambas direcciones. En la comunicación sin conexión se usa el protocolo UDP. Los datos se envían y reciben en paquetes separados e individuales (*datagramas*). La entrega no está garantizada, pueden ser duplicados, perdidos o llegar en un orden distinto al que se envió. Cada vez que se quiere enviar información se vuelve a establecer contacto con el servidor.

3.2.1.2 Configuración del socket

El primer paso para la construcción de una aplicación con sockets es determinar la función que va a desempeñar, para seleccionar el tipo de comunicación y el protocolo, con estos datos se puede construir el descriptor del socket invocando la función `socket()`.

```
descriptor =socket(familia_de_protocolos,
            tipo_de_comunicación,
            protocolo);
```

Para poder iniciar la comunicación en red, se deben configurar las direcciones locales y remotas del socket, invocando otras funciones del API dependiendo del uso que se le va a dar. La Tabla 4 enlista las funciones que se deben usar en cada situación.

Tabla 4 Funciones de configuración de sockets

Comunicación	Uso del socket	Información local	Información remota
Con conexión	Servidor	`bind`	`listen y accept`
	Cliente	`connect`	`connect`
Sin conexión	Servidor	`bind`	`recvfrom`
	Cliente	`bind`	`sendto`

3.2.1.3 Envío y recepción de datos

Las funciones para el envío y recepción de datos se listan en la Tabla 5, cual usar depende del *tipo de comunicación* que se haya configurado en el socket.

Tabla 5 Funciones para el envío y recepción de datos

Comunicación	Transmisión	Recepción	Descripción
Con conexión	`send`	`recv`	Se puede controlar el comportamiento del socket por medio de banderas.
	`write`	`read`	Utiliza un buffer de datos simple.
	`writev`	`readv`	Utiliza bloques de memoria no contiguos como buffer de datos

Comunicación	Transmisión	Recepción	Descripción
Sin conexión	sendto	recvfrom	Utiliza un buffer de mensajes simple
	sendmsg	recvmsg	Utiliza una estructura de mensajes simple como buffer.

3.2.2 Programación del servidor

El propósito de la construcción de un programa servidor es para que atienda las solicitudes de un cliente. Para que el socket pueda recibir estas solicitudes se le debe indicar a la capa de transporte cual es la dirección IP y número de puerto del protocolo que se usará para recibir datos. La interfaz proporciona la instrucción `bind()` para registrar y especificar que está en uso la dirección y puerto indicado. Esta función se aplica para ambos tipos de comunicación.

Servidor orientado a conexión

Después de que se ha unido el socket a un puerto de protocolo local se debe invocar la función `listen()` para colocar al servidor en modo de atención pasiva. El socket envía un mensaje de confirmación al solicitante indicando que se recibió su solicitud de conexión. Para aceptar y establecer la conexión se debe invocar la función `accept()`, esta bloquea el flujo de ejecución del servidor hasta que se reciba una solicitud del cliente. Una vez que se ha establecido la conexión se puede iniciar un ciclo de envío y recepción de información.

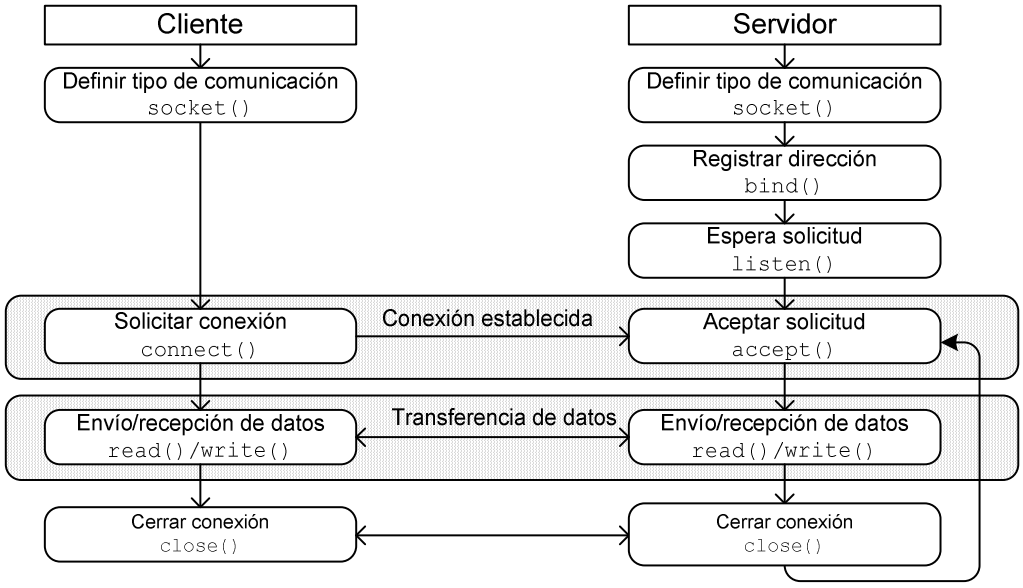


Figura 21 Sockets orientados a conexión

La Figura 21 muestra la secuencia en que se deben realizar las llamadas a funciones en un entorno de sockets que usan protocolos orientados a conexión. Este es el esquema de operación que se usará para desarrollar el protocolo de comunicación para la teleoperación de los robots CNC.

3.2.3 Programación del cliente

En la programación de un cliente orientado a conexión no es necesario averiguar su dirección local, por lo tanto no se requiere usar la función `bind()`. La conexión con el servidor se establece con la invocación de la función `connect()`. El envío y recepción de información se debe realizar usando las funciones que correspondan al tipo de comunicación.

3.3 Open GL

OpenGL (Open Graphics Library) es una interfaz de programación para el hardware de gráficos. La interfaz consiste en un conjunto de funciones que permiten al programador especificar los objetos y operaciones necesarias para producir gráficos tridimensionales de alta calidad [35].

OpenGL no incluye comandos de alto nivel para modelar objetos complejos. Incluye un conjunto de primitivas geométricas básicas que se usan para construir objetos más complejos, puntos, líneas y polígonos.

3.3.1 Sintaxis

La sintaxis de las funciones se muestra en la Figura 22, comienzan por las letras "*gl*" en minúscula, para indicar que es una función OpenGL, seguida del nombre del comando con la primera letra en mayúscula. El número indica la cantidad de argumentos que espera la función y el último carácter el tipo de dato de los parámetros.

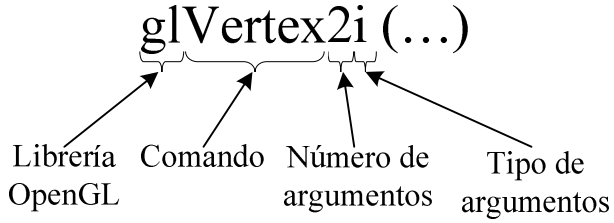


Figura 22 Sintaxis de las funciones OpenGL

En la Tabla 6 se muestran las letras utilizadas como sufijos para indicar el tipo de dato de los parámetros de cada función.

Tabla 6 Sufijo de las funciones de OpenGL

Sufijos	Tipos de datos	Definición en lenguaje C	Definición de tipo
`b`	Entero de 8 bits	single char	`GLbyte`
`s`	Entero de 16 bits	short	`GLshort`
`i`	Entero de 32 bits	long	`GLint, GLsizei`
`f`	Real 32 bits	float	`GLfloat, GLclampf`
`d`	Real 64 bits	double	`GLdouble, GLclampd`
`ub`	Entero positivo 8 bits	unsigned char	`GLubyte, GLboolean`
`us`	Entero positivo 16 bits	unsigned short	`GLushort`
`ui`	Entero positivo 32 bits	unsigned long	`GLuint`

Algunas funciones de OpenGL utilizan el sufijo "v" para indicar que sus parámetros se reciben en un arreglo que contiene los datos, en lugar de varios argumentos individuales. Ej.

```
glColor3f(1.0,1.0,1.0);
```

Es equivalente a:

```
float color[] = {1.0,1.0,1.0};
glColor3fv(color);
```

OpenGL define la constante GLvoid en lugar de "void" del lenguaje C. Las constantes definidas en OpenGL se declaran en mayúsculas, las palabras se unen con guión bajo.

3.3.2 Máquina de estados

OpenGL funciona como una máquina de estados, es decir si a una propiedad se le asigna un valor determinado, todo lo que se haga a partir de ese momento se verá afectado por ese valor, hasta que éste se modifique o desactive de forma explicita.

Algunas variables de estado se activan con el comando `glEnable()` y se desactivan con el comando `glDisable()`. Cada variable de estado o modo tiene un valor por defecto y existen también funciones para conocer el valor actual de esa variable. Cuando se necesita almacenar el estado actual de la escena, se pueden almacenar las variables de estado en una pila, con el comando `glPushAttrib()` y restaurar el estado anterior con `glPopAttrib()`.

3.3.3 Paradigma begin-end

Se usa en la construcción de los objetos geométricos. Los objetos se definen a partir de un conjunto de vértices. Para que OpenGL pueda procesar correctamente este conjunto de vértices, se deben agrupar entre las funciones `glBegin()` y `glEnd()`. La función `glBegin()` recibe un parámetro que le indica que tipo de objeto va a construir, el número de vértices debe ser consistente con el parámetro.

El siguiente programa define un conjunto de vértices para dibujar un polígono, el último vértice se une al primer vértice automáticamente como se observa en la Figura 23.

```
glBegin(GL_POLYGON);
      glVertex2f(0.0, 0.0);
      glVertex2f(0.0, 3.0);
      glVertex2f(4.0, 3.0);
      glVertex2f(6.0, 1.5);
      glVertex2f(4.0, 0.0);
glEnd();
```

Resultado obtenido con los parámetros GL_POINTS y GL_POLYGON.

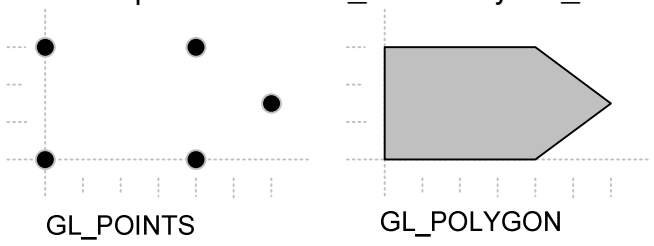


Figura 23 Paradigma glBegin-glEnd para la definición de vértices

3.3.4 Visualización

Para la creación de los gráficos en 3D, OpenGL implementa una tubería de graficación integrada por tres matrices de transformación. Cada vértice pasa por esta tubería. La coordenada original se multiplica por las matrices de visualización y modelo, a continuación se realiza un proceso de recorte, si el vértice queda dentro del área a presentar, se multiplica por la matriz de puerto de visión en el que se mapean las coordenadas normalizadas a coordenadas en la ventana de la pantalla.

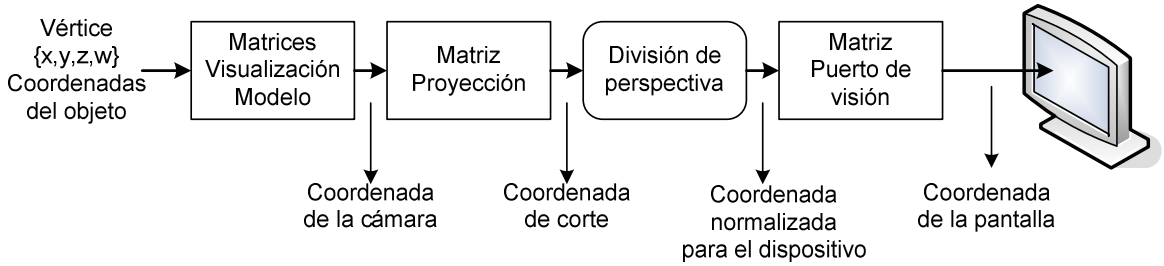


Figura 24 Tubería de graficación OpenGL

3.3.4.1 Matriz de visualización y modelo

Se encarga de crear la abstracción de posicionar y orientar una *cámara* en el mismo espacio que el escenario. Para obtener la visión adecuada se debe mover la cámara de su posición por default, a una que sea más adecuada para ver el escenario, realizando operaciones de rotación, traslación o escala. Estas operaciones forman parte de la matriz de visualización y modelo.

`glMatrixMode(GL_MODELVIEW);`	Activar la matriz de visualización y modelo
`glLoadIdentity();`	Iniciar con la matriz unidad para limpiar cálculos previos.
`gluLookAt(ojo.x,ojo.y,ojo.z,` `ver.x,ver.y,ver.z);`	Posición y orientación de la cámara

3.3.4.2 Matriz de proyección

Usar la matriz de proyección es análogo a escoger el lente o zoom de una *cámara*. Especifica el volumen de la escena, algunos objetos quedarán fuera y serán recortados. La cámara en su posición inicial, se ubica en el origen del eje de coordenadas, viendo hacia el lado negativo del eje z (Figura 25).

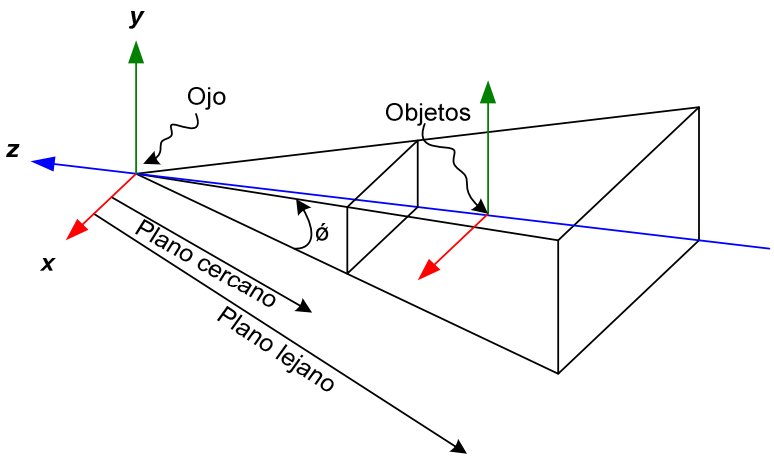


Figura 25 Posición inicial de la cámara

3.3.4.3 Agregar volumen a la visualización

La configuración de la matriz de proyección se realiza con la función `gluPerspective()`. El parámetro "***anguloVision"*** indicado en la Figura 25 como ø, especifica el ángulo (grados) de apertura de la pirámide. Un ángulo muy abierto permite tener una *visión más amplia* e incluir muchos más objetos. El parámetro "***aspecto"*** indica la relación existente entre las medidas del plano *x* y el plano *y*. El parámetro **cerca** es la distancia entre la posición de la cámara y el plano más cercano, el parámetro **lejos** es la distancia entre la posición de la cámara y el plano más lejano, ambos deben ser positivos.

`glMatrixMode(GL_PROJECTION);`	Activar la matriz de proyección
`glLoadIdentity();`	Iniciar con la matriz unidad para limpiar cálculos previos.
`gluPerspective(anguloVision, aspecto, cerca, lejos);`	Configuración del volumen o proyección

3.3.4.4 Matriz puerto de visión

El puerto de visión indica la forma y tamaño del área disponible en la pantalla para dibujar la escena. Los vértices obtenidos de la tubería de transformación, se multiplican por esta matriz para obtener las coordenadas de pantalla. Es el último paso para dibujar la escena. La configuración de la matriz de puerto de visión se realiza con la función `glViewport()`. Los parámetros ***inicio*** y ***fin,*** especifican la esquina inferior izquierda del puerto de visión, los valores por default son 0,0. Los parámetros ancho y alto, especifican el tamaño de la ventana asignada a OpenGL.

`glViewport(inicio,fin,ancho,alto);`	Configuración de la matriz puerto de visión

3.4 Control Numérico Computarizado (CNC)

Se considera control numérico computarizado al dispositivo capaz de controlar la posición y velocidad de los motores que accionan los ejes de una máquina, por medio de instrucciones en forma numérica y que son enviados desde una computadora. Los movimientos se realizan de forma automática a partir de estos comandos. Gracias a esto, se pueden realizar movimientos complejos. La computadora es capaz de mover los tres (o más) ejes simultáneamente, logrando realizar líneas diagonales, círculos y cortes tridimensionales

Estos robots se encuentran disponibles en una amplia variedad de tipos, tamaños y aplicaciones. Las más comunes son: taladros, tornos y fresadoras [23], aunque también se usan para barrenar, perforar, desgastar, pulir, soldar, cortadoras de flama, ensamblado de componentes electrónicos, equipos láser, pruebas automáticas, fotografía, artesanías, etc.

3.4.1 Objetivos de una CNC

Construir de piezas complejas, mayor precisión, reducción de tiempo y costos. Estos objetivos se logran gracias a la reducción en el tiempo de programación, incremento en las capacidades operacionales y lograr que el proceso de fabricación sea más amigable con el usuario.

3.4.2 Clasificación de sistemas de CNC

De acuerdo a [28] se dividen fundamentalmente en:

1. **Sistemas de Control punto a punto y contorneo abierto**. Este tipo de control no proporciona información de retroalimentación a la consola de control y prácticamente ya no existen en la actualidad.

2. **Sistemas de Control de trayectoria continua (Continuos-Path Control).** Contiene elementos computacionales (interpolators) que permiten el cálculo de puntos sucesivos en segmentos de línea o curvas, usando poca información de entrada, como coordenada final, radio y coordenada del centro. La interpolación puede ser linear, circular o parabólica. Los Sistemas de Control de trayectoria continua también se les denominan control numérico de contorneo (Contouring Controls).

3. **Sistemas de Control numérico de contorneo cerrado.** Se basan en la posición y regeneración de la velocidad. Para controlar el comportamiento dinámico y la posición final del deslizamiento de la máquina, se apoyan en una variedad de transductores de posición tanto analógicos como digitales.

La Figura 26 muestra los componentes básicos de un sistema de contorneo cerrado.

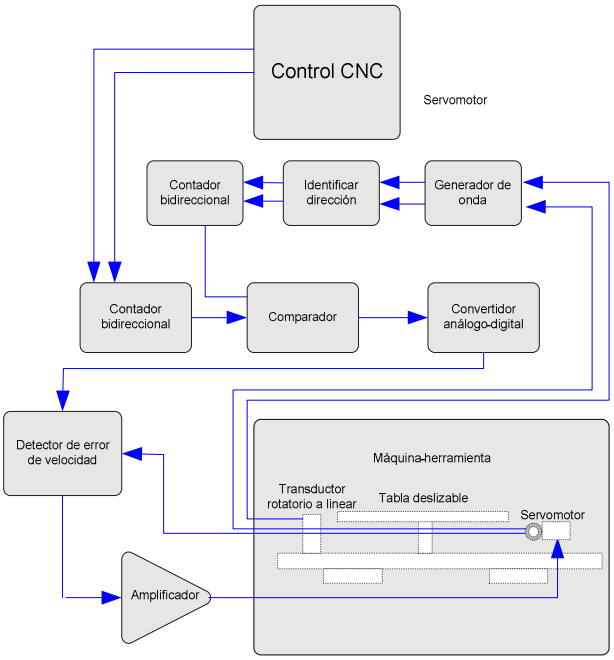


Figura 26 Componentes básicos de un sistema de contorneo cerrado

3.4.3 Componentes principales de una CNC

Los componentes principales de un sistema CNC se pueden ver en la Figura 27. Consiste en una base rígida con una tabla de trabajo horizontal movible, un huso vertical para sostener la herramienta, selector automático de herramienta y la unidad de control, que puede ser una computadora o un microcontrolador.

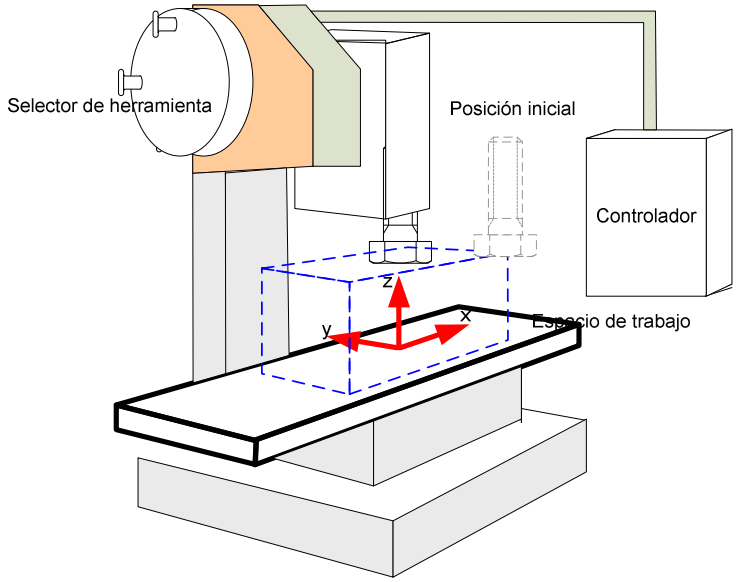


Figura 27 Componentes básicos de un sistema CNC

3.4.3.1 Componentes mecánicos

La mesa de trabajo está montada en dos barras deslizables que le permiten el movimiento del objeto, en dirección horizontal ortogonal. El soporte vertical de la herramienta también está montado en una barra deslizable para producir el movimiento vertical y generar un tercer movimiento ortogonal. Los robots se clasifican de acuerdo al número de ejes de control, las fresadoras normalmente tiene de tres a cinco ejes, mientras que los tornos normalmente tienen dos ejes.

El movimiento de la tabla de trabajo a lo largo de la barra deslizable se controla por medio de un tornillo sinfín con una tuerca que circula sobre un cojinete, como se ve en la Figura 28 Entre el rodamiento y la tuerca existe un espacio de precisión que les permite el movimiento. Los rodamientos circulan a lo largo de la cuerda y se mueven a través del paso interior en la tuerca.

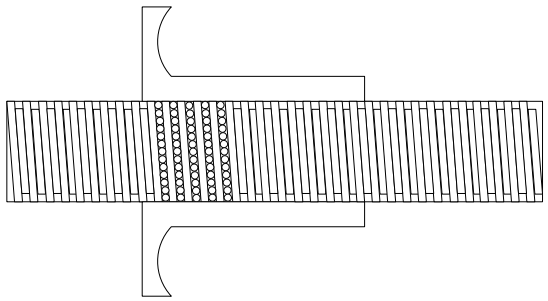


Figura 28 Tornillo sinfín y tuerca con rodamiento.

3.4.3.2 Movimiento y control de los motores

Para mover los tornillos sinfín se emplean comúnmente dos tipos de motores: Servomotores de corriente directa y motores a pasos.

Los servomotores son los más usados para aplicaciones de control numérico. Tienen gran potencia y velocidad, con baja inercia y operación suave. Se usan en conjunto con codificadores circulares que son montados en el eje. Los codificadores se usan como información de retroalimentación para conocer la posición del motor.

Los motores a pasos son relativamente baratos, con una gran potencia de torsión a baja velocidad y se requieren controles electrónicos más simples que los servomotores. Operan en base a la transmisión de pulsos. El número total de pulsos determina la cantidad de rotación y por lo tanto la distancia recorrida, mientras la frecuencia de pulsos determina la velocidad de movimientos.

En la Figura 29 se muestra un diagrama del detalle de montaje de la mesa de trabajo, tornillo sinfín, tuerca con cojinetes y el servomotor.

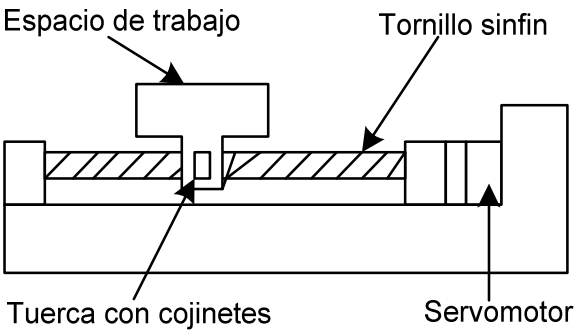


Figura 29 Montaje de la mesa de trabajo

3.4.4 Movimientos y medidas de los ejes

La base para especificar los movimientos de los ejes de los sistemas CNC es el sistema de coordenadas rectangulares. Todas las posiciones son descritas en términos de distancias a partir del origen (X0.000, Y0.000, Z0.000). Las medidas se realizan perpendicularmente a lo largo de los ejes (Figura 30). Las unidades de medida se pueden expresar en pulgadas o milímetros.

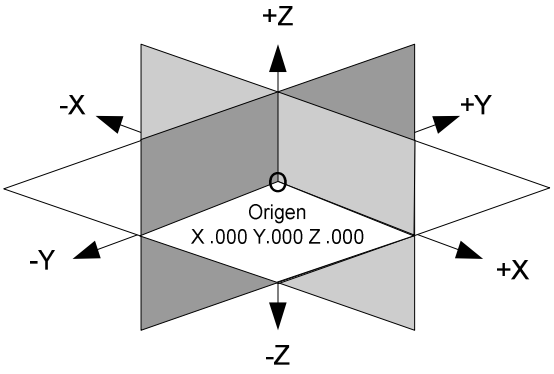


Figura 30 Sistema cartesiano usado en los sistemas CNC

3.4.5 Asignación de los ejes de la máquina

Las direcciones de movimiento que se muestran en la Figura 31 son los que normalmente se usan en una fresadora. El movimiento más largo que pueda realizar la herramienta normalmente se ajusta a lo largo o paralelo al eje X. Los movimientos de corte se realizan en una sola dirección, por lo general hacia la derecha desde una vista frontal. A este se denomina "*X positivo (+X)*" y el movimiento en la dirección opuesta "*X negativo (-X)*". Partiendo del origen en un ángulo de 90º sobre el eje X, se define al eje Y con dirección positiva y negativa relativa a la herramienta de corte. El eje Z se usa para mover la herramienta de corte.

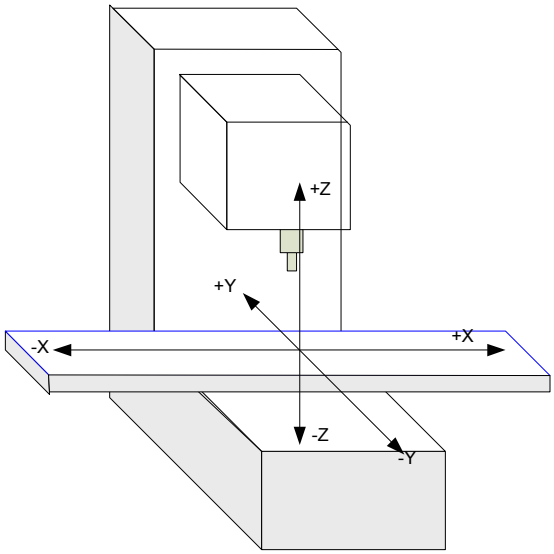


Figura 31 Orientación de los ejes cartesianos en una máquina CNC vertical

3.4.6 Programación

La mayoría de sistemas CNC usan el estándar RS274-D como lenguaje de programación, proporciona la flexibilidad de integrar ambientes CAD/CAM, CNC y otros sistemas, permite definir de forma precisa lo que se desea obtener. Existen otros lenguajes de programación.

3.4.6.1 RS274-D G-Code

Obtiene el nombre de **G-Code** porque las funciones básicas empiezan con la letra G aunque su nombre estándar es **RS274-D**. El lenguaje incluye los elementos para definir los diferentes aspectos de control de una máquina CNC, define movimientos rápidos (G00), lineales (G01), circulares (G02 y G03), controlar el radio de velocidad (códigos F), unidades de medidas entre milímetros y pulgadas, compensación de longitud a partir del tamaño de la herramienta (G43) y funciones preprogramadas (G65), velocidad de corte (comandos S), selección de herramientas (comandos T) y funciones auxiliares. En el *anexo A* se muestra la gramática para implementar un intérprete de comandos.

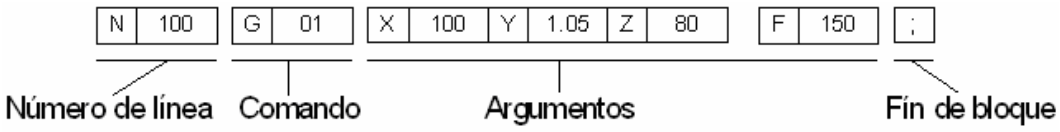
3.4.6.2 Bloques

El lenguaje RS274-D se codifica en líneas de código a las que se les denomina "**bloques**" o líneas de información. Cada bloque incluye comandos para indicarle a la máquina CNC realizar diferentes tareas. Los programas se guardan en archivos de texto.

Una línea de información, se estructura de la siguiente forma:

- Inicia opcionalmente con un número de línea seguido de uno o más comandos
- Un comando consiste de una letra seguida de un número entero.

- Opcionalmente un comando puede ir acompañado de uno o más argumentos.
- Un argumento consiste en una letra seguida de un número entero o real.
- Opcionalmente se puede definir un carácter que indique el fin de bloque.



Esta línea se interpreta de la siguiente forma: en el bloque 100, la herramienta se debe mover en línea recta (G01) desde su posición actual hacia la coordenada determinada sobre los ejes X 100 mm, Y 1.05 mm y en Z 80 mm, a una velocidad constante de 150 mm/min (F150).

Número de línea. Una línea es una letra N seguida de un número entero sin signo, entre 0 y 99999. Los números pueden repetirse o ir desordenados, los intérpretes normalmente las leen en el orden de aparición y se ignoran. Este es un parámetro opcional, pero si se usan se deben escribir correctamente.

Comando. Es una letra G,M,S seguida de un número entero, los números de línea no son comandos.

Parámetro. Es una letra distinta a N, G, M o S seguidas de un valor entero o real, algunos parámetros tienen interpretaciones diferentes, dependiendo del comando al cual están asociados.

En la Tabla 7, se muestra la descripción de los comandos y parámetros existentes en el lenguaje RS274-D.

Tabla 7 Parámetros y comandos del lenguaje RS274-D

Comando	Descripción
D	Radio de compensación de la herramienta
F	Velocidad constante, en milímetros/minuto o pulgadas/minuto
G	Función general
H	Índice para seleccionar compensación de herramienta
I	Distancia a lo largo del eje X para arcos
J	Distancia a lo largo del eje Y para arcos
K	Distancia a lo largo del eje Z para arcos
L	Número de repeticiones
M	Funciones misceláneas
N	Número de línea
P	Tiempo de retardo en milisegundos para G04
Q	Velocidad de incremento en ciclos G83
R	Radio de un arco, menor de 180 grados
S	Velocidad de corte
T	Seleccionar herramienta

Comando	Descripción
X	Coordenada en el eje X Tiempo de retardo en segundos para G04
Y	Coordenada en el eje Y
Z	Coordenada en el eje Z

3.4.6.3 Comentarios

Los comentarios pueden definirse de dos formas: Resto de línea y Multilínea. En el área indicada para comentarios se permite escribir cualquier carácter imprimible.

Resto de línea, se agrega el carácter '#' en cualquier posición de una línea, el texto que le siga es ignorado hasta el final de la línea.

Multilínea, se definen comentarios en bloque, mediante la combinación de apertura **'(' o '(*'** y cerradura de caracteres **'*)' o ')'**.

3.4.6.4 Códigos G

En la siguiente tabla se listan los comandos G.

Tabla 8 Listado de códigos G

G-Code	Descripción	Parámetros
G00	Movimiento rápido	`G00 X__ Y__ Z__`
G01	Interpolación lineal	`G01 X__ Y__ Z__ F__`
G02	Interpolación circular en sentido del reloj	`G02 X__ Y__ R__ F__` `G02 X__ Y__ I__ J__ F__`
G03	Interpolación circular en contrasentido del reloj	`G03 X__ Y__ R__ F__` `G03 X__ Y__ I__ J__ F__`
G04	Tiempo de espera	`G04 X__ ; (segundos)` `G04 P__ ; (milisegundos)`
G17	Seleccionar el plano X-Y	`G17`
G18	Seleccionar el plano X-Z	`G18`
G19	Seleccionar el plano Y-Z	`G19`
G20	Seleccionar unidad de medida en pulgadas	`G20`
G21	Seleccionar unidad de medida en milímetros	`G21`
G22	Habilitar coordenadas de límite de movimiento	`G22 X_Y_Z_I_J_K_`
G23	Deshabilitar coordenadas de límite de movimiento	`G23`
G28	Ir a posición Inicio	`G28 X__ Y__ Z__`
G29	Volver de posición Inicio	`G29 X__ Y__ Z__`
G40	Cancelar compensación de radio de herramienta.	`G40`
G41	Compensación izquierda de radio de herramienta.	`G41 D__`
G42	Compensación derecha de radio de herramienta.	`G42 D__`
G43	Compensación de longitud positiva de herramienta.	`G43 H__ ; G43 Z__ H__`
G44	Compensación de longitud negativa de herramienta.	`G44 H__ ; G44 Z__ H__`
G49	Cancelar compensación de longitud de herramienta.	`G49`

G-Code	Descripción	Parámetros
G50	Desactivar operación de escalado	G50
G51	Activar operación de escalado	G51 I__ J__ K__ P__
G54, G55, G56, G57, G58, G59	Seleccionar sistema de coordenadas de trabajo	G54 G55 G56 G57 G58 G59
G90	Coordenadas absolutas	G90
G91	Coordenadas relativas	G91
G92	Programar la coordenada del cero absoluto	G92 X__ Y__ Z__

3.5 Puertos paralelos

Un puerto es un conjunto de líneas de comunicación que usa el microprocesador o CPU para intercambiar datos con los otros componentes[29]. El puerto paralelo se diseñó específicamente para la comunicación con impresoras, pero puede ser usado como un puerto de entrada/salida de datos con cualquier dispositivo externo. Un puerto paralelo es un medio ampliamente usado para enviar las instrucciones de control a un robot CNC.

3.5.1 Puerto paralelo estándar

La versión original del puerto paralelo, tiene un total de 12 bits de salida y 5 bits de entrada, accesibles vía tres puertos consecutivos en el espacio de direcciones de entrada/salida del procesador, comúnmente en los siguientes rangos:

Tabla 9 Direccionamiento del puerto paralelo

Nombre	Base/Datos	Estado	Control
LPT1	3BCh	3BDh	3BEh
LPT2	378h	379h	37Ah
LPT3	278h	279h	27Ah

La primer columna es el nombre común del puerto para la plataforma Windows, la segunda indica la dirección base del puerto, también recibe el nombre de registro datos o simplemente la dirección del puerto. La tercer columna es el registro de estado y la cuarta columna es el registro de control.

La conexión física al puerto paralelo se realiza por medio de un conector tipo DB25, en la Figura 32 se muestra la asignación de cada pin a los registros de direccionamiento.

3.5.2 Programación del puerto paralelo

El puerto paralelo responde a 2 instrucciones de salida por medio de los registros de datos y control; 3 instrucciones de entrada por los registros de datos, estado y control (Tabla 10). Dos de las instrucciones de entrada permiten leer la información enviada con las instrucciones de salida. La instrucción de entrada en

el registro de estado sirve para obtener información en tiempo real generada por el dispositivo externo.

Los registros pueden ser referenciados de la siguiente forma:

Tabla 10 Programación del puerto paralelo

Registro	Dirección	Pines / Bits								Instrucción	
		7	6	5	4	3	2	1	0		
Datos	Base + 0	9	8	7	6	5	4	3	2	Salida	Entrada
Estado	Base + 1	-11	10	12	13	15					Entrada
Control	Base + 2					-17	16	-14	-1	Salida	Entrada

La dirección base puede ser: 278/378/3BC Hex, dependiendo del número de puerto. Algunos bits se interpretan de forma invertida.

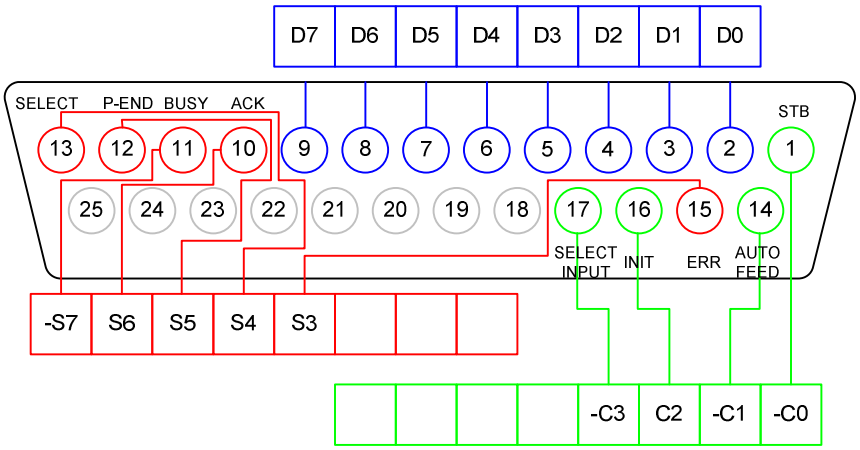


Figura 32 Interfaz del puerto paralelo

3.6 Motores a pasos

Los motores de paso a paso (PaP) son dispositivos electromagnéticos, rotativos, incrementales que convierten pulsos digitales en rotación mecánica [6], la característica principal de estos motores es el hecho de poder moverlos un paso a la vez por cada pulso que se le aplique. Este paso puede variar desde 90° hasta 1.8°[37].

La cantidad de rotación es directamente proporcional al número de pulsos y la velocidad de rotación es relativa a la frecuencia de dichos pulsos .Los motores de paso a paso son simples de operar en una configuración de lazo cerrado, debido a su tamaño proporcionan un excelente torque a baja velocidad.

3.6.1 Tipos de motores

Las bobinas (fases) de los motores PaP pueden ser de imán permanente, reluctancia variable e híbridos. Los motores de imán permanente se dividen en bipolares y unipolares.

3.6.2 Control de los motores a pasos

Para realizar el control de los motores paso a paso, es necesario generar una secuencia determinada de impulsos, estos deben ser capaces de entregar la corriente necesaria para que las bobinas del motor se exciten. La Figura 33 muestra un diagrama de bloques de un sistema con motores paso a paso.

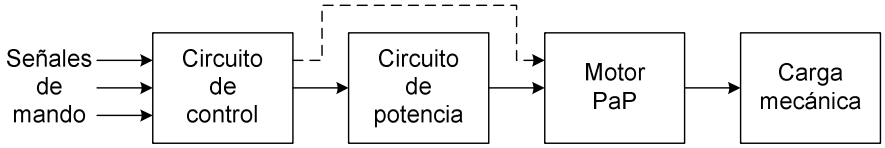


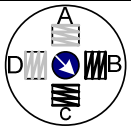
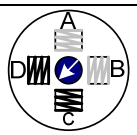
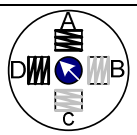
Figura 33 Diagrama de bloques de un sistema con motor paso a paso

Secuencias para manejar motores paso a paso unipolares

Existen tres secuencias posibles para este tipo de motores [24][52], todas se repiten desde el principio al llegar el último paso. Para *revertir* el sentido de giro, se deben ejecutar las secuencias en modo inverso.

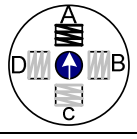
Secuencia completa. Con esta secuencia el motor avanza un paso por vez y debido a que siempre hay al menos dos bobinas activadas, se obtiene un alto torque de paso y de retención.

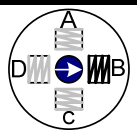
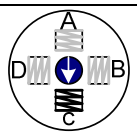
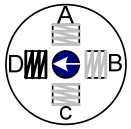
Tabla 11 Secuencia completa de un motor a pasos

Paso	Bobina A	Bobina B	Bobina C	Bobina D	
1	**Encendida**	**Encendida**	Apagada	Apagada	
2	Apagada	**Encendida**	**Encendida**	Apagada	
3	Apagada	Apagada	**Encendida**	**Encendida**	
4	**Encendida**	Apagada	Apagada	**Encendida**	

Secuencia suave. En esta secuencia se activa solo una bobina a la vez. En algunos motores esto brinda un funcionamiento más suave. La desventaja es que al estar activada solo una bobina, el torque de paso y retención es menor.

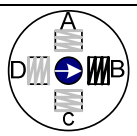
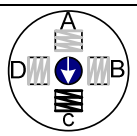
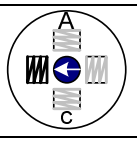
Tabla 12 Secuencia suave de un motor a pasos

Paso	Bobina A	Bobina B	Bobina C	Bobina D	
1	**Encendida**	Apagada	Apagada	Apagada	

Paso	Bobina A	Bobina B	Bobina C	Bobina D	
2	Apagada	**Encendida**	Apagada	Apagada	
3	Apagada	Apagada	**Encendida**	Apagada	
4	Apagada	Apagada	Apagada	**Encendida**	

Secuencia de medio paso: En esta secuencia se activan las bobinas de tal forma que permitan un movimiento igual a la mitad del paso real. Para ello se activan primero 2 bobinas y luego solo 1 y así sucesivamente. La secuencia completa consta de 8 movimientos en lugar de 4.

Tabla 13 Secuencia de medio paso de un motor a pasos

Paso	Bobina A	Bobina B	Bobina C	Bobina D	
1	**Encendida**	Apagada	Apagada	Apagada	
2	**Encendida**	**Encendida**	Apagada	Apagada	
3	Apagada	**Encendida**	Apagada	Apagada	
4	Apagada	**Encendida**	**Encendida**	Apagada	
5	Apagada	Apagada	**Encendida**	Apagada	
6	Apagada	Apagada	**Encendida**	**Encendida**	
7	Apagada	Apagada	Apagada	**Encendida**	

Paso	Bobina A	Bobina B	Bobina C	Bobina D	
8	**Encendida**	Apagada	Apagada	**Encendida**	

Resumen

Se han explicado las tecnologías que se usarán en la etapa de implementación (Capítulo 5) de los objetivos objetivos específicos 2,3,4 y 5. Para la construcción del protocolo de teleoperación, se revisó la forma de construir aplicaciones cliente servidor por medio de sockets. Se presentó la forma de desarrollar aplicaciones en OpenGL para la construcción del módulo de simulación de la aplicación de escritorio, se explicó ampliamente la forma de operación del lenguaje RS274-D conocido también como G-Code y el funcionamiento de los robots CNC. Finalmente, para la construcción del controlador del robot MAXNC-10, se revisó la forma de trabajar de los motores a pasos y el puerto paralelo.

En el siguiente capítulo se realiza el análisis de requerimientos, construcción de la arquitectura, casos de uso y diseño de clases del modelo para la Teleoperación de robots CNC que corresponden a los objetivos específicos 1,2 y 3.

Capítulo 4 Modelo propuesto para la teleoperación de robots CNC

4.1 Introducción

En este capítulo se realiza el análisis y diseño del modelo que atiende los objetivos específicos 1,2 y 3, planteados en la sección 1.1.2.

El diseño de la solución propuesta se basa en el *Paradigma Orientado a Objetos* [4], se desarrolla siguiendo un subconjunto de los pasos especificados en el *Proceso Unificado de Desarrollo*[27] aplicables al dominio del problema que se está analizando. Los diagramas se presentan en el estándar del *Lenguaje Unificado de Modelado*[50]. En la construcción de la arquitectura se ha privilegiado la incorporación de *Patrones de Diseño*[18] con el objetivo de permitir la reutilización más eficiente del modelo propuesto y lograr la integración congruente de sus componentes.

4.2 Descripción general

De acuerdo al planteamiento del problema expuesto en la sección 1.3, se presenta el esquema de funcionamiento esperado, en el cual participa un *usuario* que se encuentra físicamente separado del robot, un *operador* encargado de acomodar las piezas a moldear, puede ser una persona o algún otro robot, la comunicación entre el cliente y el servidor se realiza por medio del protocolo de aplicación propuesto en este trabajo (Figura 34).

El *usuario* ingresa al módulo cliente y se registra en el sistema, si el robot está disponible, se le habilitarán las opciones para el envío de comandos para la teleoperación. En caso contrario el sistema habilitará únicamente los módulos de edición y simulación.

Para la teleoperación del robot, el *usuario* debe contar con un programa en el formato RS274-D, conectarse al servidor y enviarlo para su ejecución. Si el

servidor se encuentra procesando otra solicitud, la petición del cliente será rechazada, hasta que se termine de atender la petición anterior, en caso contrario el servidor recibirá la secuencia de comandos y esperará la instrucción para iniciar su procesamiento. En el lado del servidor habrá un *operador*, encargado de acomodar las piezas que se van a modelar antes de iniciar la ejecución de los comandos. Una vez terminada la ejecución, el servidor notificará al cliente.

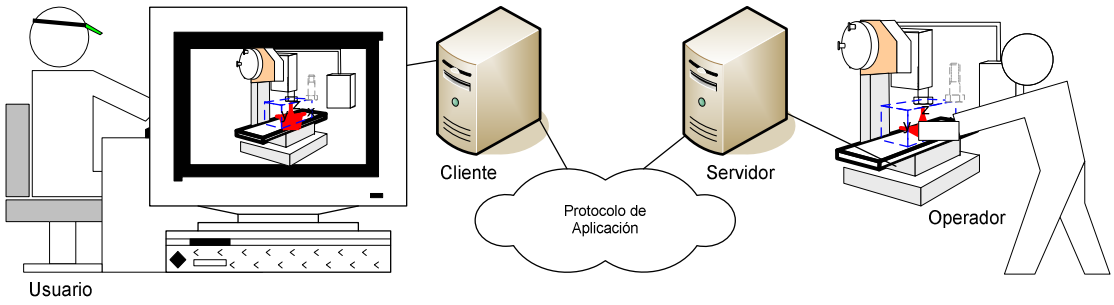


Figura 34 Modelo conceptual

4.3 Requerimientos

El objetivo de esta sección es definir la funcionalidad del sistema, delimitar los alcances, detallar los aspectos técnicos a considerar y definir la interfaz con el usuario.

Un diagrama de *casos de uso* es un conjunto de escenarios que describen la interacción entre los usuarios y el sistema. Los elementos de un *diagrama* de casos de uso son los casos de uso, actores y relaciones (Figura 35).

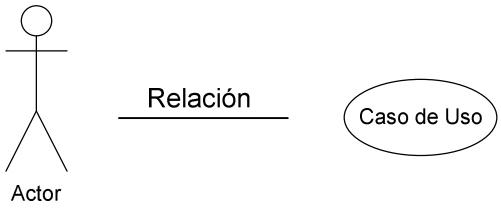


Figura 35 Diagrama de caso de uso

El *actor* representa a un usuario o algún otro sistema que interactúa con el sistema que se está modelando. Un *caso de uso* es una vista externa del sistema, representa un conjunto de acciones que se deben seguir para completar una tarea. La *relación* sirve para unir al actor y el caso de uso con el que interactúa.

4.3.1 Requerimientos funcionales

Desde el punto de vista del usuario, el sistema debe cubrir los requerimientos de funcionalidad (Figura 36). **Edición y simulación.** El sistema debe contar con una herramienta que permita abrir, guardar y actualizar archivos de comandos RS274-D. Ver la simulación de los programas de comandos en diferentes perspectivas, realizar los ajustes que sean necesarios en el código hasta obtener el resultado

esperado. **Teleoperación.** El sistema debe ofrecer la infraestructura que permita la interacción a distancia entre el usuario y el robot, quienes pueden estar separados físicamente pero conectados mediante la red de Internet. **Control**. Implementar los algoritmos requeridos para interpretar y procesar los comandos de control, enviar las instrucciones requeridas para realizar el movimiento de los motores, realizar ajustes a la posición inicial del robot.

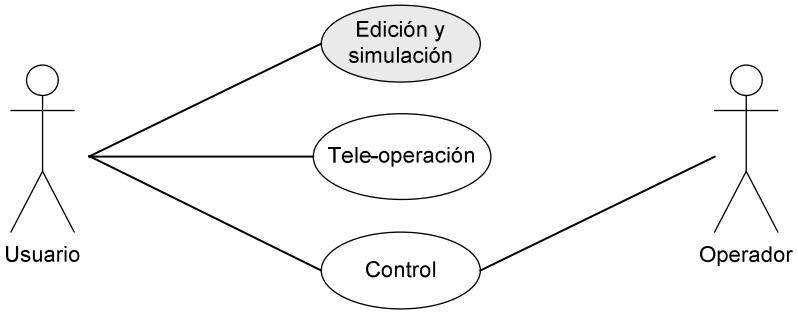


Figura 36 Requerimientos funcionales del usuario

4.3.2 Requerimientos no funcionales

4.3.2.1 Usabilidad.

La solución está orientada a usuarios de diferentes niveles de dominio de las herramientas de programación. Muchas universidades, centros de investigación e industrias tienen la necesidad de operar robots CNC casi de forma inmediata y no disponen de los recursos para invertir en un largo proceso de capacitación o adquisición de licencias de software complementario.

4.3.2.2 Resolución y precisión.

Los robots CNC operan en resoluciones determinadas por los componentes usados en su fabricación. Se debe permitir la especificación de la resolución en que opera el robot. La precisión de los movimientos se determinará en función de la resolución del robot.

El sistema debe permitir el uso de unidades en pulgadas y milímetros de acuerdo al estándar RS274-D (*sección 3.4.6*). Los algoritmos se implementarán usando como unidad de medida las pulgadas.

4.3.2.3 Soporte.

El diseño de la solución se regirá por estándares de análisis, diseño y programación, que permitan al usuario final realizar mantenimiento y actualización del modelo.

4.3.2.4 Condiciones de diseño.

La incorporación de nuevos tipos de robots no debe alterar el diseño general del modelo presentado.

4.3.2.5 Interfaces de hardware.

El control de los robots se realiza vía el puerto paralelo, la secuencia de pasos para controlar el robot se implementa en un controlador.

4.3.2.6 Interfaces de red.

El sistema debe permitir la operación vía Internet, para lo cual el protocolo a usar en la implementación será el protocolo de transporte TCP, porque garantiza la entrega oportuna y confiable de paquetes.

4.4 Análisis

El diseño de la arquitectura depende de las condiciones especificadas en los requerimientos funcionales, los requerimientos no funcionales relacionados con la tecnología serán considerados durante la etapa de *implementación* en la sección 5.2.2. El requerimiento de *Edición y simulación* corresponde al uso del Framework en un caso particular, por lo cual no se incluye en esta etapa de análisis y diseño, su explicación se realiza en el capítulo de *implementación* en la sección 5.2.5.

4.4.1 Arquitectura

El diseño de la arquitectura se apoya en el patrón de diseño arquitectónico en capas propuesta por [21]. El modelo presenta diferentes niveles de abstracción (Figura 37). La *capa de control* está compuesta por los algoritmos de control físico del robot y el protocolo de transporte de red. La *capa de componentes* implementa los algoritmos de alto nivel, para el control de los robots CNC y el protocolo de aplicación. El *Framework* constituye una capa que permite la integración de los componentes de una forma ordenada, estructurada y sencilla. El usuario desarrollará soluciones concretas en la *capa de aplicación* a partir de la especialización del Framework. El control de robots particulares se realizará mediante la implementación de los drivers de control del robot.

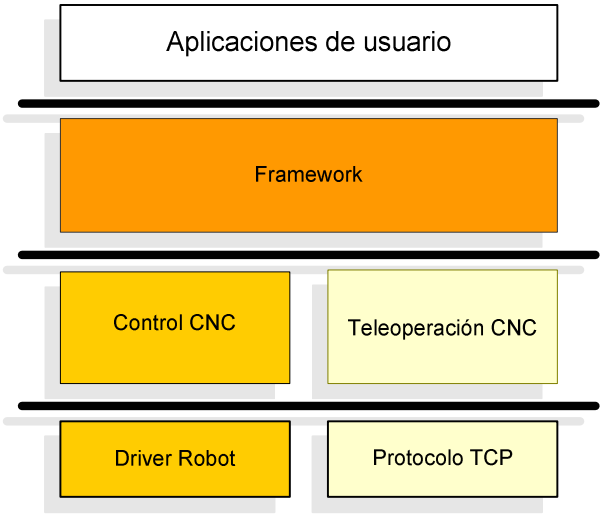


Figura 37 Modelo en capas

Los componentes se estructuran de forma lógica de acuerdo a la función que realizan, el diagrama que se muestra en la Figura 38 permite observar el flujo de ejecución entre los diferentes subcomponentes.

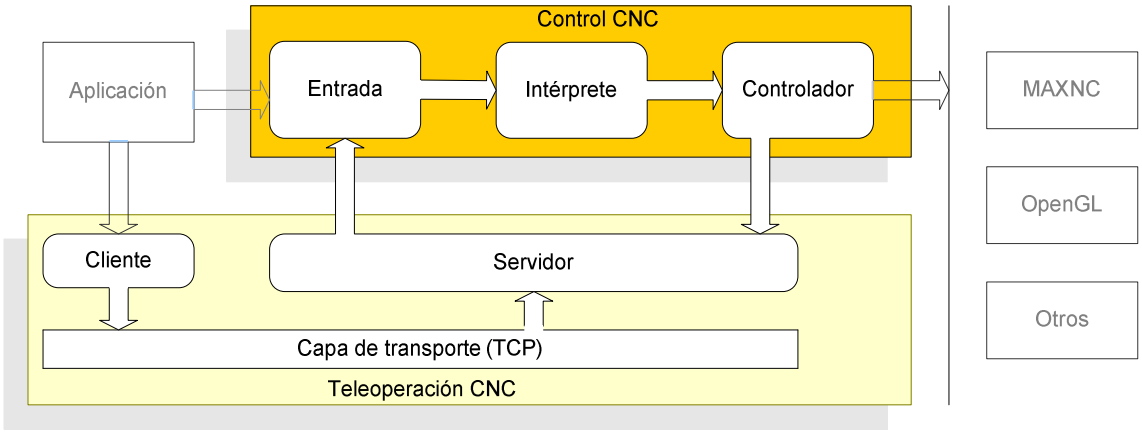


Figura 38 Arquitectura

4.4.1.1 Framework

El objetivo de este componente es facilitar el desarrollo de nuevas aplicaciones de control de robots CNC, su función es ocultar en una interfaz común los detalles de implementación del componente de control y el de teleoperación.

4.4.1.2 Control CNC

Control CNC. Este componente está integrado por tres elementos que permiten enviar instrucciones a una máquina CNC física o virtual. Es responsable de recibir el archivo o flujo de datos con las instrucciones en formato de códigos RS274-D, enviarlas al intérprete, para que éste genere las instrucciones de bajo nivel para el movimiento de los motores o para la simulación del programa.

Entrada

Recibe la secuencia de comandos que será procesada por el sistema, la entrada de datos puede ser desde un archivo, un flujo, red, etc. Este componente especifica la forma en que se deben enviar los datos. Almacena las instrucciones sin realizar validación de sintaxis.

Intérprete

Recibe una secuencia de comandos desde la *entrada* de datos, se encarga de realizar un análisis de los comandos, verificar que el código es correcto y convertirlo a una estructura de datos del tipo lista simplemente enlazada para que pueda ser ejecutado.

Controlador

Realiza la traducción de la lista de comandos a instrucciones de bajo nivel. Mediante un *método plantilla* [18] define la estructura de los algoritmos que implementan las instrucciones del lenguaje RS274-D. Ofrece puntos de extensión,

que permiten su adaptación a diferentes configuraciones de robots físicos o virtuales.

4.4.1.3 Teleoperación CNC

Provee la infraestructura de comunicación entre cliente y servidor, proporciona funciones específicas para la operación a distancia de los robots CNC, es independiente del control y configuración del robot. Se comunica al componente de Control CNC en el lado del servidor.

Protocolo de aplicación

Es un protocolo de comunicación ubicado en la capa de aplicación del modelo OSI elaborado ex profeso para este trabajo. Está formado por un conjunto de instrucciones para realizar la conexión, envío y ejecución de comandos entre el cliente y servidor.

Cliente

Se implementa como un cliente ligero, no realiza procesamiento sobre los comandos a enviar o recibidos, no incluye elementos de la capa de presentación. El programador será el responsable de desarrollar una interfaz gráfica independiente del protocolo, que servirá para ser operado por el usuario final. Aunque podría ser operado desde un cliente telnet.

Servidor

Actúa como un dispositivo de ***entrada*** para el componente Control CNC. Se diseña como un *decorador* [18] de las funciones del framework. Implementa funciones específicas de comunicación, conexión, recepción de comandos y monitoreo.

4.4.2 Vista de casos de uso

El conjunto de actores y casos de uso que se describen a continuación, detallan los alcances concretos del sistema. Este modelado elimina cualquier ambigüedad, que se pudiera presentar para el cumplimiento de los objetivos de este trabajo. Se desprende de los casos de uso y la arquitectura presentada previamente. Explican la forma de operar el sistema en cada tarea.

Los diagramas de secuencia de los casos de uso, se pueden consultar en el *Anexo B*, en estos se puede observar el envío de mensajes entre clases.

4.4.2.1 Definición de Actores

A continuación se listan los actores que interactúan con el sistema:

4.4.2.1.1 Usuario

Es la persona encargada de usar el programa cliente, para realizar la operación a distancia del robot CNC. Puede realizar el trabajo de editar y simular los programas RS274-D (G-Code). Se ubica del lado del cliente.

4.4.2.1.2 Operador

Es la persona o dispositivo encargado de colocar las piezas a moldear, en la tabla o área de corte del robot CNC. Tiene contacto directo con el robot. Se ubica del lado del servidor.

4.4.2.2 Casos de uso del sub módulo: Control CNC

Las funciones requeridas para el control del robot, se muestran en la Figura 39, como se puede apreciar, en algunos casos se requiere la participación del *usuario* y *operador*. Los casos de uso ocultan la operación del robot a distancia.

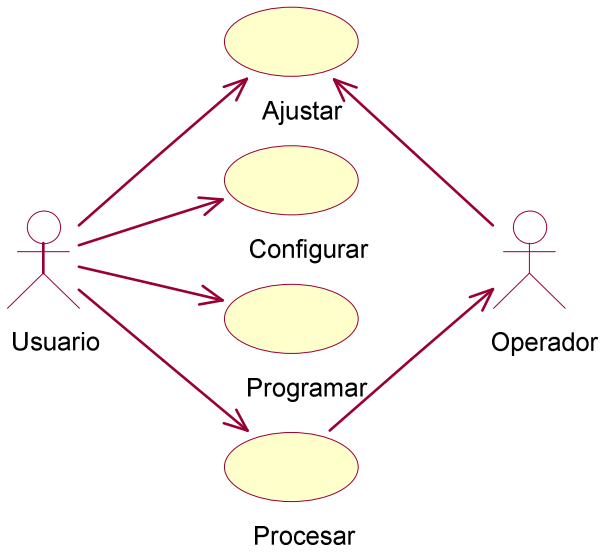


Figura 39 Casos de uso del sub módulo: Control CNC

4.4.2.2.1 Ajustar

Objetivo	*Establecer la posición X=0, Y=0, Z=0 del robot, antes de iniciar el modelado de las piezas.*
Actor participante	*Usuario* *Operador*
Precondiciones	• *El operador ha ingresado al sistema.*

Flujo de eventos

Actor	Sistema servidor
1.- El usuario activa la función Ajustar.	*2.- Despliega el cuadro de diálogo con las opciones para ajustar los ejes X,Y,Z. Solicita la posición {0,0,0}, indicando una coordenada, o un modo de paso a paso.*
3.-El operador, proporciona la coordenada o la cantidad de pasos.	*4.- Calcula los movimientos requeridos y los envía al controlador.* *5.- Establece la posición actual del robot en {0,0,0}.*
6.- El operador notifica al usuario que el ajuste ha sido realizado. Termina el caso de uso.	

Poscondiciones	• *El robot se encuentra en la posición {0,0,0}.*

4.4.2.2.2 Configurar

Objetivo	*Definir parámetros generales de operación del robot.*
Actor participante	*Usuario*
Precondiciones	• *El operador ha ingresado al sistema.*

Flujo de eventos	Actor	Sistema servidor
	1.- El usuario activa el módulo Configurar.	*2.- Despliega el cuadro de diálogo con los parámetros requeridos.*
	3.-El operador, proporciona la información solicitada.	*4.- Se envía la información al controlador del robot y se re-inicializa con los nuevos parámetros.*
		5.- Termina el caso de uso.

Poscondiciones	• *El robot ha sido reconfigurado.*

4.4.2.2.3 Programar

Objetivo	*Preparar el programa de comandos para su envío al robot.*
Actor participante	*Usuario*
Precondiciones	• *El operador ha ingresado al sistema.*

Flujo de eventos	Actor	Sistema servidor
	1.- El usuario activa la función Nuevo o Abrir.	*2.- Se habilita el editor de comandos.*
	3.-El operador escribe los comandos o los abre de un archivo.	*4.- Los comandos se agregan al componente de Entrada.*
	5.- El operador agrega otros comandos.	*6.- Solicita el análisis del archivo.*
		7.- Continúa con el paso 5, hasta que obtenga el programa correcto. Termina el caso de uso.

Poscondiciones	• *Se obtiene un programa de comandos en formato RS274-D.*

4.4.2.2.4 Procesar

Objetivo	*Ejecutar el programa de comandos RS274-D.*
Actor participante	*Usuario*
Precondiciones	• *El robot ha sido ajustado.* • *El sistema tiene un programa de comandos*

Flujo de eventos	Actor	Sistema servidor
	1.- El usuario activa la función Ejecución.	*2.- Envía los comandos al controlador.*
		3.- Se realiza la ejecución de un comando preparado en la lista simplemente enlazada.
		4.- Verificar si se ha solicitado la interrupción de la ejecución o es el final de la secuencia de comandos, en caso afirmativo ir a 5, para el caso contrario regresar a 3.
		5.- Terminar de procesar secuencia de comandos.

Poscondiciones	• *Se obtiene el modelo físico del programa proporcionado.*

4.4.2.3 Casos de uso del sub módulo: Teleoperación CNC

En la Figura 40 se muestran los casos de uso relacionados con el módulo de Teleoperación. Para el *usuario* representan funciones ofrecidas por el protocolo de comunicación. Para el *operador* son funciones que permitirán iniciar o detener el servidor. Las funciones de la aplicación se especifican en ***NEGRITAS CURSIVAS***, los comandos del protocolo se indican en texto en *CURSIVA*.

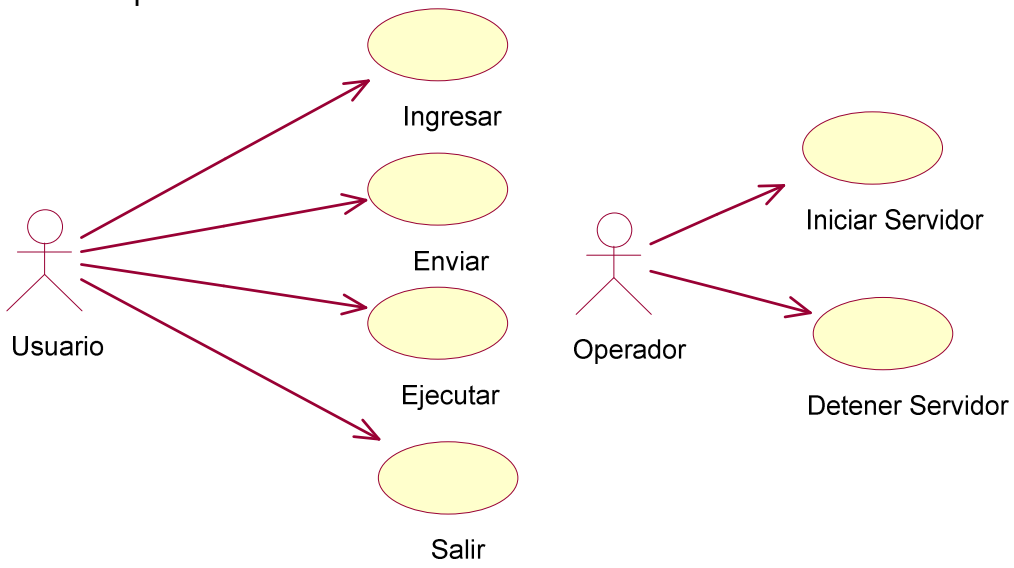


Figura 40 Casos de uso del sub módulo: Teleoperación CNC

4.4.2.3.1 Caso de uso: Ingresar

Objetivo	***Establecer conexión entre cliente y servidor***		
Actor participante	*Usuario*		
Precondiciones	• *El usuario ha ingresado al sistema cliente.* • *El usuario no ha establecido conexión con el servidor.* • *El servidor está recibiendo solicitudes.*		
Flujo de eventos	Actor	Sistema cliente	Sistema servidor
	*1.- El usuario activa la función "**INGRESAR**", desde la aplicación cliente.*	*2.- Responde mostrando un cuadro de diálogo, para solicitar la dirección y puerto del servidor.*	
	3.-Proporciona los datos solicitados.	*4.-Abre una conexión con el servidor. Si no se puede establecer el contacto con el servidor, se ejecuta 4.1*	*5.- Establece la conexión con el cliente. Espera el comando de conexión.*
		*6.- Envía el comando "**INGRESAR** <usuario>" para, establecer una sesión de trabajo con el servidor*	*7.- El sistema registra la dirección IP del cliente y responde con el comando "**ACEPTADO**".*

			*8.- Se habilitan las opciones de **ENVIAR, PROCESAR y SALIR** en el programa cliente. Termina el caso de uso.*

Flujo alternativo *4.1 Si el servidor no está recibiendo solicitudes, Se deshabilitan las opciones de **ENVIAR, PROCESAR y SALIR** en el programa cliente. Termina el caso de uso.*

Poscondiciones
- *El usuario es **ACEPTADO** en el sistema servidor.*

4.4.2.3.2 Caso de uso: Enviar

Objetivo ***Transferir los comandos RS274-D del programa cliente al servidor***

Actor participante *Usuario*

Precondiciones
- *El usuario ha sido **ACEPTADO** en el sistema servidor.*
- *El usuario tiene una secuencia de comandos RS274-D, en una estructura de bytes.*

Flujo de eventos

Actor	Sistema cliente	Sistema servidor
*1.- El usuario activa la función "**ENVIAR_ NUEVO**" o **"ENVIAR AGREGAR"**, desde la aplicación cliente.*	*2.- El sistema convierte los comandos en el formato del editor, a la secuencia de bytes esperada por el servidor.*	
	*3.-Construye la instrucción "**ENVIAR_NUEVO**" ó "**ENVIAR_AGREGAR**", según corresponda, junto con la secuencia de bytes.*	*4.- Recibe la secuencia de comandos y la almacena en el componente Entrada. Contesta al cliente con la instrucción "**RECIBIDO**".*
	5.- El caso de uso termina.	

Poscondiciones
- *Se habilita la opción **EJECUTAR** en el sistema cliente.*

4.4.2.3.3 Caso de uso: Ejecutar

Objetivo *Indicar al servidor que realice la ejecución de los comandos enviados.*

Actor participante *Usuario*

Precondiciones
- *El sistema cliente tiene habilitada la opción **EJECUTAR**.*

Flujo de eventos

Actor	Sistema cliente	Sistema servidor
*1.- El usuario activa la función "**EJECUTAR**", desde la aplicación cliente.*	*2.-Envía la instrucción "**EJECUTAR**". Deshabilita la opción "**ENVIAR**".*	*3.- Recibe el comando, verifica si el robot, se encuentra procesando alguna otra solicitud, si el robot está disponible solicita la ejecución de los comandos disponibles en el objeto ENTRADA.*

			*Terminada la ejecución, envía el comando **TERMINADO**. Si el robot no está disponible, se ejecuta el flujo alternativo 3.1.*
		*4.- Deshabilita la opción ejecutar, habilita la opción **ENVIAR**, para una nueva secuencia de comandos.* *5.- El caso de uso termina.*	

Flujo alternativo *3.1 Si el robot no esta disponible, el servidor envía el comando "**ERROR_EJECUTAR**", termina el caso de uso. En el cliente quedan habilitadas las opciones de ENVIAR y EJECUTAR.*

Poscondiciones
- *Se deshabilita la opción **EJECUTAR** en el sistema cliente.*

4.4.2.3.4 Caso de uso: Salir

Objetivo *Cerrar la conexión entre cliente y servidor.*

Actor participante *Usuario*

Precondiciones
- *El usuario ha establecido conexión con el servidor.*
- *El usuario ha sido **ACEPTADO** en el sistema servidor.*

Flujo de eventos

Actor	Sistema cliente	Sistema servidor
1.- El usuario activa la función "Salir", desde la aplicación cliente.	*2.- Envía el comando "**SALIR**".*	*3.- El sistema elimina de su registro la dirección IP del cliente y responde con el comando "**ADIOS**".*
	*5.- Se deshabilitan las opciones de **ENVIAR, PROCESAR y SALIR** en el programa cliente. Termina el caso de uso.*	*4.- Cierra la conexión con el cliente.*

Poscondiciones
- *El sistema tiene habilitada la opción INGRESAR.*

4.4.2.3.5 Caso de uso: Iniciar Servidor

Objetivo ***Habilitar el servidor para recibir solicitudes.***

Actor participante *Operador*

Precondiciones
- *El operador ha ingresado al sistema servidor.*
- *Se ha realizado la inicialización de un robot.*

Flujo de eventos

Actor	Sistema servidor
1.- El operador activa la función "Iniciar Servidor", desde la aplicación servidor.	*2.- Responde mostrando un cuadro de diálogo, para solicitar el puerto del servidor.*
3.-Proporciona los datos solicitados.	*4.- Habilitar el servidor para recibir solicitudes.* *5.- Habilitar la opción "Detener Servidor". Termina el caso de uso*

Poscondiciones	• *El servidor está en modo de recepción de solicitudes.*

4.4.2.3.6 Caso de uso: Detener Servidor

Objetivo	*Deshabilitar el servidor para recibir solicitudes.*	
Actor participante	*Operador*	
Precondiciones	• *El servidor está en modo de recepción de solicitudes.*	
Flujo de eventos	Actor	Sistema servidor
	1.- El operador activa la función "Detener Servidor", desde la aplicación servidor.	*2.- Envía el comando "**DESCONEXION**" a todos los clientes conectados.* *3.- Cerrar las conexiones.* *4.- Habilitar la opción "Iniciar Servidor".Terminar caso de uso.*
Poscondiciones	• *Habilitar la opción "Iniciar Servidor".*	

4.5 Diseño

Se presentan las clases de acuerdo a los dos componentes de la arquitectura, describen las relaciones lógicas entre los componentes de la arquitectura. Especifica los puntos que deben ser extendidos para la construcción de una solución específica.

4.5.1 Diagramas de clases

4.5.1.1 GFramework

Propósito.
Es una clase integradora de los componentes de Teleoperación y Control, ofrece una interfaz para el desarrollado de aplicaciones. Corresponde al componente Framework de la arquitectura.

Diseño.
El diseño general de la clase se basa en el patrón *Mediador* [18], facilita la comunicación entre las diferentes clases de la arquitectura, por medio de una interfaz común.

El constructor de la clase recibe como parámetro un objeto derivado de la clase `GAbstractDriver(GConcreteDriver)` construido desde la aplicación cliente, lo que se basa en el *patrón estrategia* [18], para seleccionar los algoritmos adecuados para cada tipo de robot.

Forma de uso.
El desarrollador de aplicaciones deberá crear instancias de esta clase, desde la aplicación específica, pasando como parámetros un objeto derivado de la clase `GAbstractDriver`, `GAbstractDebug` y el número de puerto en que se inicializará el servidor. La funcionalidad de la aplicación se obtendrá mediante la invocación de los métodos de ésta clase.

4.5.1.2 Diagrama de clases del sub módulo: Control CNC

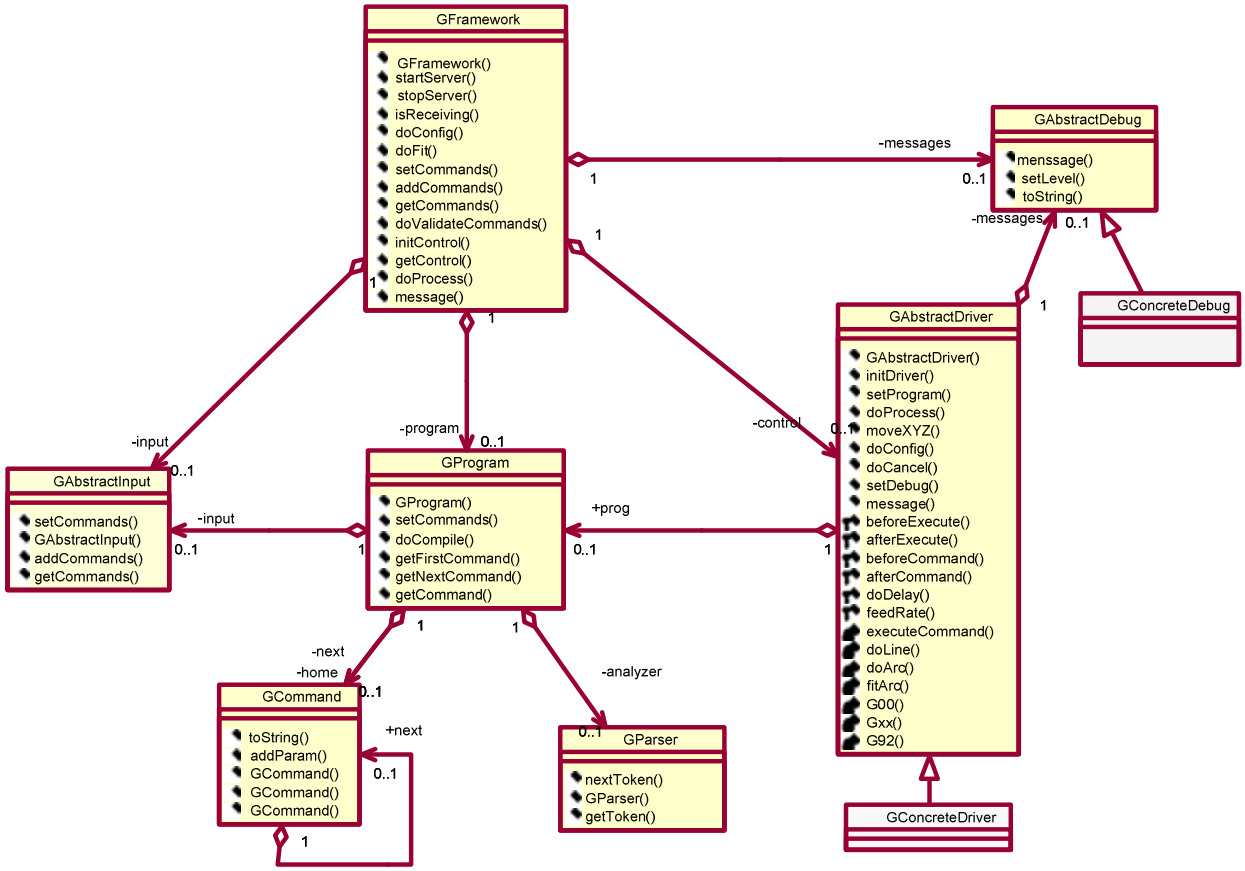


Figura 41 Diagrama de clases del sub modulo: Control CNC

4.5.1.2.1 GAbstractInput

Propósito.
Es la clase encargada de inicializar, recibir y agregar los comandos que serán enviadas al robot.

Diseño.
Corresponde al componente *entrada* de la arquitectura, sirve como el único canal de recepción de los comandos, que serán procesados por el modelo.

Forma de uso.
El usuario no hará uso directo de esta clase, el envío de los mensajes se realizará por medio de los métodos proporcionados por el Framework.

4.5.1.2.2 GProgram

Propósito.
Implementa los algoritmos para analizar el código fuente de los programas en el formato RS274-D. Construye una estructura de datos con los comandos procesados.

Diseño.
Esta clase implementa el algoritmo para procesar los tokens obtenidos de la secuencia de comandos.

Uso.
Esta clase ofrece un método para recibir los comandos que debe procesar. La compilación se inicia mediante el método `doCompile()`. Los métodos son invocados desde el Framework.

4.5.1.2.3 GParser

Propósito.
Implementa el analizador sintáctico de la secuencia de comandos en formato RS274-D, proporciona los métodos requeridos para obtener e indicar el tipo y valor de cada token.

Diseño.
La clase se integra a la clase `GProgram`, por el mecanismo de composición.

Uso.
Esta clase es usada por la clase `GProgram`.

4.5.1.2.4 GCommand

Propósito.
Es la estructura de datos responsable de almacenar la versión *compilada* de la secuencia de comandos RS274-D.

Diseño.
Se implementa mediante una *lista simplemente enlazada*.

Uso.
Esta clase es usada por la clase `GProgram`.

4.5.1.2.5 GAbstractDriver

Propósito
Define la interfaz que debe ser implementada por el programador para el soporte de nuevos tipos de robots. Implementa los algoritmos de control del lenguaje RS274-D.

Diseño.
El diseño general de la clase forma parte del patrón estrategia que fué definida en el contexto del Framework, los métodos `execute()` y `executeCommand()`, se han implementado como un *método plantilla* [18]. Los algoritmos generales para procesar los comandos RS274-D se implementan en esta clase. Se han definido los métodos `beforeExecute()`, `afterExecute()`, `beforeCommand()` y

`afterCommand()`, que deben ser implementados en las clases concretas, si quieren agregar un comportamiento adicional en los robots.

Uso.
El programador debe crear una clase concreta para el tipo de robot que quiere controlar, debe implementar el método `moveXYZ`, que será usado por el Framework en el contexto del patrón estrategia.

4.5.1.3 Diagrama de clases del sub módulo: Teleoperación CNC

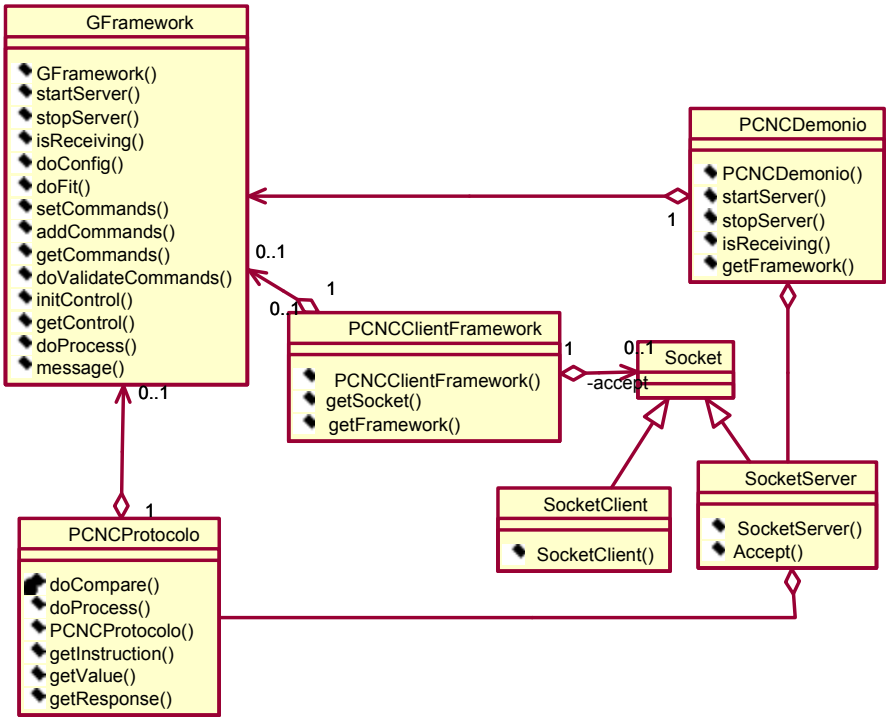


Figura 42 Diagrama de clases del sub módulo: Teleoperación CNC

4.5.1.3.1 PCNCDemonio

Propósito
Inicia un hilo de ejecución responsable de atender las solicitudes de conexión de un cliente. Cada solicitud es recibida por el hilo padre pero procesada por un hilo independiente.

Diseño
Se integra al Framework por composición, desconoce las funciones propias de los sockets o del protocolo de comunicación, su función es iniciar el servidor. La interfaz pública, incluye los métodos `startServer()` y `stopServer()` responsables de iniciar y detener el servidor.

Uso
Para iniciar o detener el servidor, se enviarán los mensajes `startServer()` y `stopServer()` desde el Framework.

4.5.1.3.2 PCNCClientFramework

Propósito
Al iniciar cada hilo de ejecución se debe establecer la relación entre el *socket* y el *Framework* que controla el robot, esta clase es responsable de establecer la relación entre ambos.

Diseño
Es un objeto compuesto por los objetos Socket y Framework.

Uso
La clase servidor creará la instancia de un objeto de esta clase por cada cliente que se conecte al servidor y será enviado al hilo de ejecución independiente, para procesar el protocolo de comunicación.

4.5.1.3.3 Socket

Propósito
Encapsular las operaciones de trabajo por medio de sockets.

Diseño
Su diseño se apoya en el concepto de herencia, las operaciones comunes para toda comunicación entre cliente y servidor se especifican en esta clase. Las operaciones específicas se delegan a la especialización de las clases derivadas.

Uso
Esta clase no se instancia de forma directa. Se obtiene una referencia a un objeto de este tipo, mediante polimorfismo.

4.5.1.3.4 SocketServer

Propósito
Implementa las operaciones propias para atender solicitudes de conexión por medio de sockets.

Diseño
SocketsServer es una clase derivada de Socket.

Uso
Se puede construir un objeto de este tipo, proporcionando el número de puerto y clientes que deberá procesar el servidor. Opcionalmente se puede indicar el método de conexión. El responsable de recibir cada solicitud de los clientes es el método `SocketServer::Accept()`, una vez que el cliente cierra la conexión, también se libera el socket.

4.5.1.3.5 SocketClient

Propósito

Implementa las operaciones propias para solicitar la conexión a un servidor por medio de sockets. Proveer los métodos para enviar y recibir datos entre cliente y servidor.

Diseño
`SocketsClient` es una clase derivada de *Socket*.

Uso
El constructor recibe como parámetros la dirección IP o nombre de dominio del servidor y el número de puerto.

4.5.1.3.6 PCNCProtocolo

Propósito
Implementa el protocolo de comunicación.

Diseño
Es un *patrón fachada* [18] de las funciones proporcionadas por el Framework, es independiente de la arquitectura del controlador, su comunicación con él, se realiza por medio de las funciones provistas por el controlador.

Uso
Es una clase utilizada por el servidor, no es necesario que sea invocada por el programador.

4.5.2 Vista de despliegue

La vista de despliegue muestra la disposición física de los distintos nodos que componen un sistema y los componentes de software sobre dichos nodos, estos no reflejan el flujo de información.

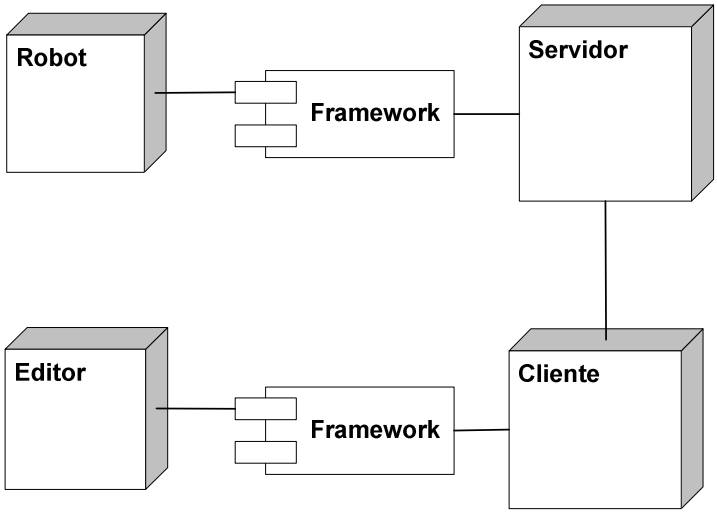


Figura 43 Diagrama de despliegue

Los componentes de hardware y software requeridos para la implementación de la arquitectura se muestran en la Figura 43. El Servidor es un equipo con el sistema operativo Windows 98 o superior, en el se instalará el Framework, el robot estará conectado al servidor por medio del puerto paralelo. El cliente puede ser cualquier computadora con el sistema operativo Windows 98 o superior, en este equipo se instalará el editor de comandos.

Resumen

En este capítulo se ha dado cumplimiento a la etapa de análisis y diseño de los objetivos específicos 1,2 y 3. Se diseñó la arquitectura del modelo propuesto, se especificaron los alcances del sistema mediante los casos de uso, el diseño del sistema se realizó por medio de los diagramas de clases y se realizó el diagrama de despliegue para conocer la ubicación de los diferentes componentes de la solución. Se han diseñado los componentes de control y teleoperación.

En el siguiente capítulo se realiza la implementación de los diseños presentados en este capítulo y su aplicación a tres casos de estudio.

Capítulo 5 Implementación y prueba del modelo para la teloperación de robots CNC

5.1 Introducción

En este capítulo se realiza la implementación del modelo diseñado en el capítulo anterior, atendiendo a la etapa de implementación de los objetivos específicos 1,2 y 3, planteados en la sección 1.1.2.

En la primera sección se realiza la implementación de la arquitectura. Posteriormente se presentan tres casos de aplicación real en los cuales se usan diferentes componentes del Framework, estos corresponden al desarrollo de los componentes de software planteados en los objetivos específicos 4 y 5 de la sección 1.1.2.

El capítulo concluye realizando pruebas de funcionamiento, tanto del Framework como de los componentes de Control y Teleoperación de robots CNC, por medio de la fabricación de dos piezas.

5.2 Implementación

5.2.1 Ambiente de desarrollo

En el capítulo 3 se presentaron las diferentes tecnologías a utilizar en la implementación del modelo propuesto. El componente de teleoperación, se implementará usando sockets. El driver para el robot virtual se desarrollará en OpenGL. Se construirá un driver para la operación del robot MAXNC-10 vía el puerto paralelo. La plataforma de desarrollo es Windows 2000. La interfaz de desarrollo de aplicaciones usado es C++ Builder, por ser entorno de programación visual orientado a eventos, proporciona un conjunto de componentes para construir una interfaz gráfica amigable con el usuario.

A continuación se explican los fragmentos más representativos de cada componente y la forma en que interactúa una aplicación con el Framework y los módulos de control y teleoperación.

5.2.2 Implementación de la arquitectura

5.2.2.1 Implementación del Framework

El Framework, se implementa en la clase `GFramework`, hace una composición de los objetos de Teleoperación y Control CNC, su función es proporcionan un conjunto de métodos *fachada*, implementa la lógica para usar las demás clases de la arquitectura.

5.2.2.2 Implementación del componente Control CNC

De acuerdo a la arquitectura descrita en 4.4.1.2, El componente de control está integrado por tres módulos: entrada, intérprete y controlador. Las clases que implementan cada uno de estos módulos y la función que realizan se puede apreciar en la Figura 44.

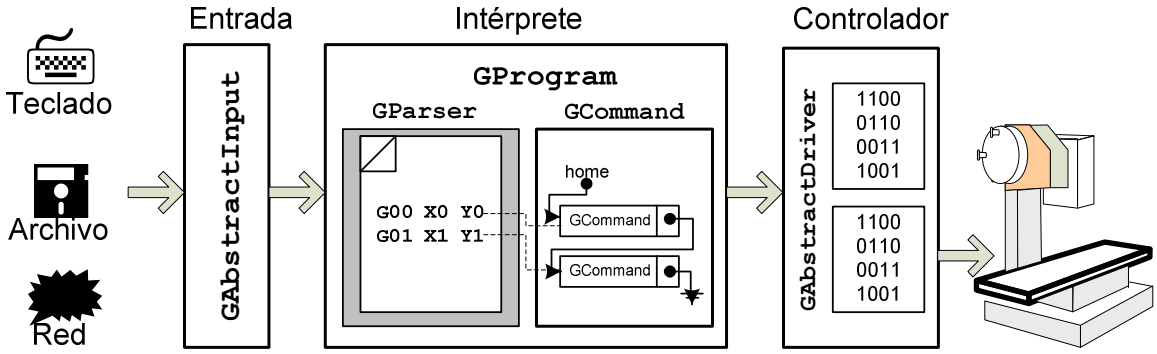


Figura 44 Función de las clases del sub módulo de control CNC

5.2.2.2.1 Implementación de la entrada abstracta

La clase `GAbstractInput` sirve de buffer para almacenar los comandos enviados desde el Framework, los comandos almacenados son usados por la clase `GProgram` para realizar el análisis sintáctico.

```
class GAbstractInput
{       private:
                std::string buffer;
        public:
                 GAbstractInput();
                 virtual void setCommands (const char *s);
                 virtual void addCommands (const char *s);
                 virtual const char *getCommands ();
};
```

5.2.2.2.2 Implementación del intérprete de comandos.

El objetivo principal de este módulo es traducir la secuencia de bytes proporcionada por la clase `GAbstractInput` a una a una estructura de datos adecuada para su ejecución. Está formado por las clases `GProgram`, encargada de recibir la secuencia de bytes en formato RS274-D, solicitar a `GParser` el análisis sintáctico de la secuencia de comandos y construir la lista simplemente enlazada con los tokens obtenidos (Figura 45). La clase `GParser`, realiza el análisis gramatical de la secuencia de bytes y lo devuelve en forma de *tokens*, se desarrolló en base a la gramática del lenguaje RS274-D que se describe en el *Anexo A.* Se construye un objeto `GCommand` por cada comando G-Code.

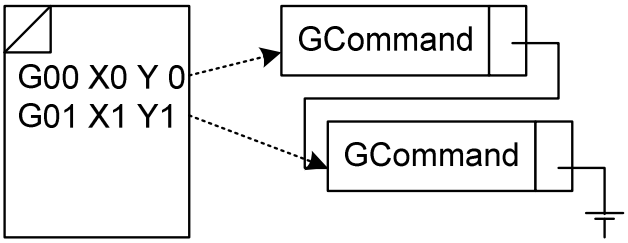


Figura 45 Compilación del lenguaje RS274-D

Diagrama de secuencia

La Figura 46 muestra el envío de mensajes entre las diferentes clases que participan en el proceso. El diagrama incluye las llamadas realizadas desde una aplicación *host* que usa el framework, y por medio de él tiene acceso a las clases proporcionadas por el módulo intérprete.

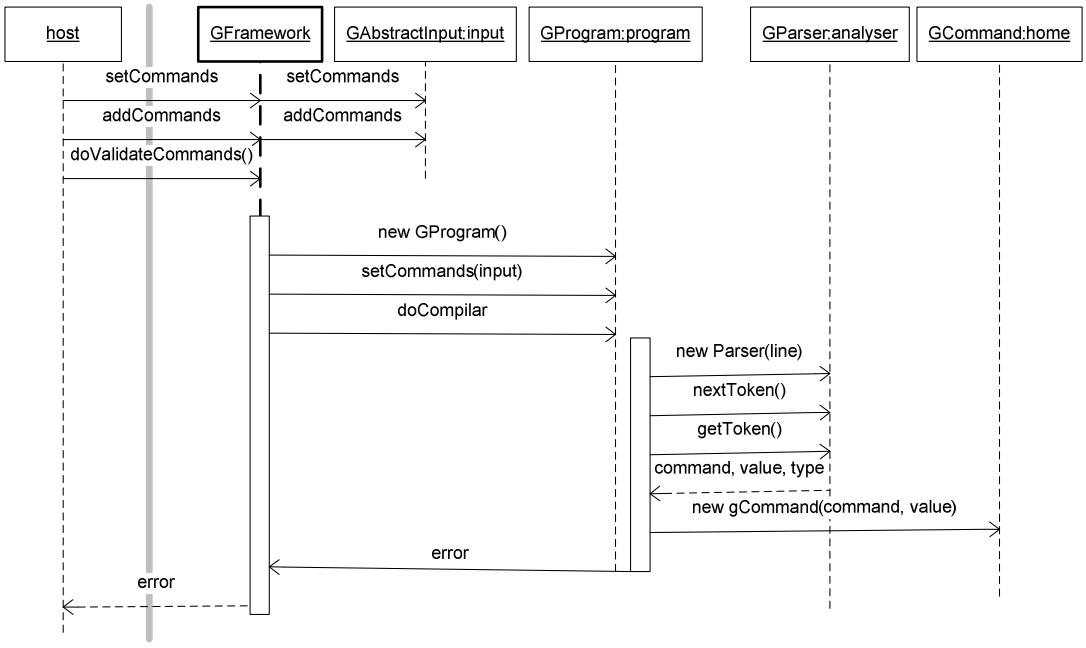


Figura 46 Diagrama de secuencia del intérprete de comandos

5.2.2.2.3 Implementación del driver abstracto.

Para propósitos de este proyecto se ha implementado en la clase `GAbstractDriver`, un subconjunto de los comandos definidos en Tabla 8, estos se listan a continuación.

Tabla 14 Comandos soportados por el Framework

G00	Movimiento rápido
G01	Interpolación lineal
G02	Interpolación circular en sentido del reloj
G03	Interpolación circular en contrasentido del reloj
G04	Tiempo de espera
G20	Seleccionar unidad de medida en pulgadas
G21	Seleccionar unidad de medida en milímetros
G28	Ir a posición Inicio, pasando por un punto
G29	Volver de posición Inicio, pasando por un punto
G90	Coordenadas absolutas
G91	Coordenadas relativas
G92	Programar la coordenada del cero absoluto

La clase `GAbstractDriver` define un conjunto de atributos con valores por defecto para su operación, estos pueden ser modificados por medio de los comandos G.

```
GAbstractDriver::GAbstractDriver(){
  ...
  H_RESOLUCION  = 1;// Este valor es redefinido en el driver concreto
  base_lpt      = 0;// Dirección del puerto paralelo
  ...
  //-- Manejar las coordenadas del robot
  actual.x     = actual.y     = actual.z     = 0;
  home.x       = home.y       = home.z       = 0;
  intermedia.x = intermedia.y = intermedia.z = 0;
  //-- Configurar parametros iniciales.
  isAbsolute    = true; // Coordenadas absolutas o relativas
  feed          = 1;    // Velocidad en revoluciones por minuto
  unidad        = inchUnit;// Unidad de medida de las coordenadas
  ...
};
```

Los valores de estos atributos pueden ser modificados de la siguiente forma.

Tabla 15 Valores por default usados por el driver abstracto

Atributo	Descripción	Default	Es modificado por
H_RESOLUCION	Especifica el número de pasos necesarios para mover un eje una distancia de una pulgada.	1	La implementación del driver concreto.
base_lpt	Especifica la dirección del puerto paralelo para enviar la secuencia de pasos.	0	El Framework recibe el valor de este parámetro desde la aplicación host.
actual.x,	La coordenada actual de la	0,0,0	Los comandos que mueven la

Atributo	Descripción	Default	Es modificado por
actual.y, actual.z	herramienta.		herramienta: G00, G01, G02, G03, G28 y G29.
home.x, home.y, home.z	Especifica la coordenada inicial de la herramienta. Las posiciones se calculan relativas a ella.	0,0,0	El comando G92 puede especificar cual es la nueva posición HOME.
intermedia.x, intermedia.y, intermedia.z	Es una coordenada usada como variable temporal.	0,0,0	Se usa y modifica por los comandos G28 (ir a HOME) y G29 (volver de HOME).
isAbsolute	Especifica el tipo de coordenadas, pueden ser absolutas(true) o relativas (false).	true	Se modifica con los comandos G90 (absolutas) y G91 (relativas).
feed	Velocidad en que se realizarán los movimientos de los ejes de la herramienta, se especifica en revoluciones por minuto.	0	La implementación del driver concreto. Y se modifica con el parámetro F de los comandos G01,G02 y G03.
unidad	Especifica la unidad de medida usada en las coordenadas, pueden ser pulgadas o milímetros, por defecto se usan pulgadas.	inchUnit	Se pueden modificar con los comandos G20 (pulgadas) y G21 (milímetros).

Resolución y precisión

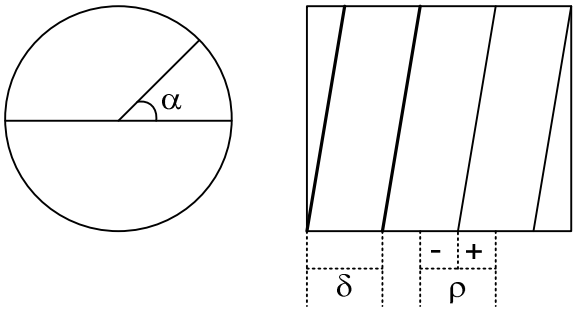
La resolución es la distancia mínima entre dos puntos que puede ser alcanzada por *un* paso del motor, está dada por la relación entre el ángulo mínimo que gira el motor y la distancia recorrida como consecuencia de ese giro.

La precisión representa un nivel de confianza, es decir, dada una posición esperada, es la distancia real obtenida, está dada por la relación entre la posición esperada y la posición obtenida. Se ve afectada como consecuencia de la resolución, condiciones físicas y mecánicas del robot y la conversión de unidades de milímetros a pulgadas. Se expresa en porcentaje de precisión.

En la clase `GAbstractDriver` se define la constante `H_RESOLUCION`, que *se usa* para especificar el número de pasos que se le deben enviar a un motor para realizar un movimiento equivalente a una pulgada. Los algoritmos de línea y de arco usan esta constante para convertir las coordenadas X,Y y Z dadas en pulgadas o milímetros a un número de pasos.

Esta constante debe ser redefinida en la implementación del driver concreto, ya que depende directamente de las características físicas del robot. El problema de la repetibilidad depender también de las condiciones físicas del robot y las variables que se consideren en la construcción del driver concreto.

El valor que debe tener `H_RESOLUCION` se calcula de la siguiente manera:



Dadas las siguientes variables:

- α Ángulo de rotación por paso, dado en grados.
- δ Distancia en pulgadas entre cuerdas del tornillo.
- ρ Precisión de la herramienta.

Se tiene:

np = 360° / α	, Número de pasos para completar una revolución.
nr = 1 / δ	, Número de revoluciones para completar una pulgada.
H_RESOLUCION = *np*nr*	, Número de pasos para realizar un movimiento equivalente a una pulgada.

La resolución que se puede obtener con estos parámetros

r = 1 / H_RESOLUCIÓN	, Distancia mínima entre dos puntos
pr= ± r / 2	, Máxima precisión en pulgadas que se puede obtener con estas resolución.

Unidades de medida

Todos los cálculos y configuraciones de la arquitectura se realizan en pulgadas, porque los componentes físicos de los robots normalmente se expresan de esta forma, sin embargo el estándar RS274-D especifica que las coordenadas también pueden ser indicadas en milímetros. La clase `GAbstractDriver` se encarga de realizar la conversión de milímetros a pulgadas cuando se usa el comando G21, antes de enviar la coordenada a los algoritmos de línea y arco.

5.2.2.2.4 Algoritmo de línea en 3D para un robot CNC.

En el conjunto de instrucciones que proporciona el lenguaje RS274-D se definen movimientos lineales por medio de los comandos G00 y G01. Los motores se mueven discretamente realizando un paso a la vez, hacia adelante, atrás o ninguno. Por esta razón se seleccionó el algoritmo de Bresenham [31] que genera coordenadas discretas de forma natural.

Este algoritmo se debe modificar por las limitaciones que tiene para ser usado en el control de robots CNC.

- No trabaja en los planos X,Y y Z.
- Los movimientos siempre se calculan en el primer cuadrante.
- No se puede seleccionar el sentido de la línea.

Algoritmo

El siguiente algoritmo sirve para dibujar líneas en los ejes XYZ, en cualquier cuadrante y sentido.

1. **Convertir las unidades de medida a número de pasos.** El programa G-Code, proporciona las coordenadas en números reales y en unidades de pulgadas o milímetros, estas deben ser convertidas a un número de pasos equivalentes a la unidad de medida. El algoritmo lee el valor de la constante `H_RESOLUCION` para realizar la conversión.

```
//-- Convertir de unidades GCODE a numero de pasos.
        ox = (int)(actual.x * H_RESOLUCION);
        oy = (int)(actual.y * H_RESOLUCION);
        oz = (int)(actual.z * H_RESOLUCION);
        dx = (int)(dest.x   * H_RESOLUCION);
        dy = (int)(dest.y   * H_RESOLUCION);
        dz = (int)(dest.z   * H_RESOLUCION);
```

2. **Calcular la distancia entre origen y destino.** Este valor se calcula para determinar cual es el eje con el mayor desplazamiento entre origen y destino. La dirección de los pasos está determinada por la diferencia entre origen y destino, este valor le indica al motor hacía donde debe girar.

$$\Delta x = x_2 - x_1$$
$$\Delta y = y_2 - y_1$$
$$\Delta z = z_2 - z_1$$

```
//-- Identificar el eje con la mayor distancia entre origen y destino
        delta_x = abs( ox - dx );   paso_x = (ox > dx ? -1 : 1 );
        delta_y = abs( oy - dy );   paso_y = (oy > dy ? -1 : 1 );
        delta_z = abs( oz - dz );   paso_z = (oz > dz ? -1 : 1 );
```

3. **Seleccionar el eje con el mayor número de pasos.** De acuerdo al algoritmo de Bresenham se selecciona el eje con la mayor distancia entre origen y destino, ya que esa será la cantidad máxima de pasos que realizará la línea. El eje seleccionado controla la secuencia de pasos, incrementando un paso en cada iteración sobre ese eje.

```
//-- Seleccionar el eje con el mayor número de pasos.
    if (delta_x>0 && delta_x >= delta_y && delta_x >= delta_z )
    {   ...
        ...
    }
    else ...
```

4. **Movimiento en los otros ejes.** El movimiento en los otros ejes está determinado por el valor de la pendiente. Sin embargo el resultado no

siempre será un valor entero. En la Figura 47, cada intersección representa el centro de una coordenada.

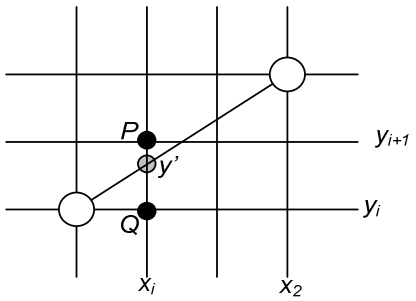


Figura 47 Movimiento en otros ejes

Para la coordenada en x_i el mejor valor de *y* es *y'*. Sin embargo como explicamos antes los movimientos son discretos. Por lo tanto se debe escoger entre:

$P = (x_i, y_i + 1)$ ó $Q = (x_i, y_i)$

De acuerdo a la ecuación de la recta:

$$y = mx + c \quad (1)$$

Se puede replantear de la siguiente manera:

$$\Delta y = m\Delta x \quad (2)$$

Para el caso de que se haya seleccionado el eje X, tenemos que Δ_x tendrá un valor de 1, por lo que Δ_y es igual a la pendiente de la recta. Si a Δ_y se le considera el *error*. Este *error* se acumula en cada iteración.

$$\varepsilon_Y = \varepsilon_Y + m \quad (3)$$

Si $\varepsilon_y > \frac{1}{2}$ entonces incrementar el valor de *y* en 1 esto es Seleccionar P y decrementar ε_y. En caso contrario seleccionar Q.

El valor de ε_y tiene un resultado de tipo flotante. Para hacer consistente el algoritmo con el uso únicamente de números enteros se usa $2\Delta x$, esto no altera su funcionamiento.

```
{     err_y=0;
      err_z=0;
      for ( int i=0; i < delta_x ; i++ )
      { do_paso_x = do_paso_y = do_paso_z = 0;
        do_paso_x = paso_x;
        err_y +=delta_2y;
        if ( err_y>= delta_x )
                {   do_paso_y = paso_y;
                    err_y-=delta_2x;
                }
        err_z +=delta_2z;
        if ( err_z>= delta_x )
                {   do_paso_z = paso_z;
```

```
                    err_z-=delta_2x;
                }
            feedRate();
            moveXYZ( do_paso_x, do_paso_y, do_paso_z);
        }
    }
```

5. **Movimiento del robot.** El algoritmo determina la dirección del *paso* y solicita que el robot realice el movimiento invocando el método `moveXYZ(paso_x, paso_y, paso_z).`

5.2.2.2.5 Algoritmos de arco en 2D para un robot CNC.

Existen dos instrucciones para dibujar un arco en el lenguaje de G-Codes, G02 y G03, la primera especifica un arco en el sentido de las mancillas del reloj, la segunda en el sentido opuesto. Las coordenadas de inicio y fin del arco también se pueden dar en dos formatos. El primero pide la coordenada destino y el radio del círculo. La segunda pide la coordenada del centro y la coordenada del final del arco. El arco se dibuja en el plano XY (Figura 48).

La construcción de un arco en 2D, se basa en el mismo principio que para el dibujado de líneas. El algoritmo de círculo de Bresenham, se debe modificar por las limitaciones que tiene para el dibujo de arcos.

- Sirve únicamente para dibujar círculos.
- Únicamente se calculan las coordenadas del primer octante.
- No se puede especificar el sentido en que se traza el círculo.
- No se permite definir las coordenadas de inicio y fin del arco.

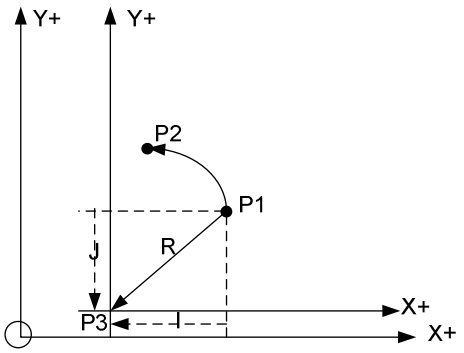


Figura 48 Arco en 2D para un robot CNC

Donde:

`P1` = Inicio del arco
`P2` = Fin del arco
`P3` = Centro del arco
`R` = Radio del arco
`I` = Distancia incremental al centro del arco a lo largo del eje X
`J` = Distancia incremental al centro del arco a lo largo del eje Y

El algoritmo que se implementó solicita la coordenada del centro del arco y la coordenada del fin del arco. El inicio del arco es la posición actual. Si los puntos de origen y destino son iguales entonces se dibuja un círculo.

1. **Convertir las unidades de medida a número de pasos.** Al igual que el algoritmo de línea, las coordenadas del programa de comandos, deben ser convertidas a número de pasos.

```
//-- Convertir de unidades GCODE a numero de pasos.
        int o_x = x0 * H_RESOLUCION;
        int o_y = y0 * H_RESOLUCION;
        int i_x = x1 * H_RESOLUCION;
        int i_y = y1 * H_RESOLUCION;
        int d_x = x2 * H_RESOLUCION;
        int d_y = y2 * H_RESOLUCION;
```

2. **Mover el centro del arco al origen del eje de coordenadas.** Para simplificar el cálculo de las coordenadas, se traslada el centro del arco al origen, esto no altera los resultados (Figura 49).

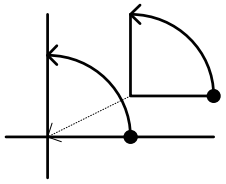


Figura 49 Mover el centro del arco al origen

```
//-- Mover el centro en el origen
      i_x =i_x - o_x;
      i_y =i_y - o_y;
      d_x =d_x - o_x;
      d_y =d_y - o_y;
```

3. **Seleccionar el sentido del giro.** Los signos del primer paso, determinarán el giro del movimiento, porque a partir de él va a continuar incrementando las posiciones hasta llegar a la coordenada final (Figura 50).

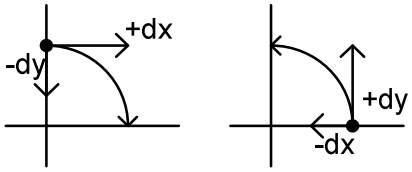


Figura 50 Seleccionar el sentido de giro

```
if (CW) { dx =  sgn(i_y); dy = -sgn(i_x);  }
else    { dx = -sgn(i_y); dy =  sgn(i_x);  }
```

4. Calcular el movimiento del siguiente paso. A diferencia del algoritmo de línea, en el caso del arco se debe ir calculando el sentido del movimiento en cada iteración. El algoritmo entrará en un ciclo, hasta que alcance la coordenada final.

```
        do {   do_paso_y=0, do_paso_x=0;
        if (i_x == 0) {
                dy = -sgn(i_y);
                dysq = (2*i_y + dy)*dy;
```

```
            }
            else if (i_y == 0) {
                   dx = -sgn(i_x);
                   dxsq = (2*i_x + dx)*dx;
            }
            ex = abs(eps + dxsq);
            ey = abs(eps + dysq);
            exy = abs(eps + dxsq + dysq);
            if (ex<ey || exy<=ey) {
                   i_x += dx;
                   eps += dxsq;
                   dxsq += 2;
                     do_paso_x = dx;
            }
            if (ey<ex || exy<=ex) {
                   i_y += dy;
                     do_paso_y = dy;
                   eps += dysq;
                   dysq += 2;
            }
             moveXYZ(do_paso_x, do_paso_y, 0 );
             feedRate();
      }   while (!(i_x==d_x && i_y==d_y));
```

5.2.2.3 Implementación del componente Teleoperación

En el *objetivo específico 3* se indica el diseño y desarrollo de un protocolo de comunicación para la teleoperación de robots CNC. Este objetivo se realiza en dos etapas: la de diseño, cubierto en la sección 4.5.1.3 y la de implementación, que se explica a continuación.

En la definición del problema se explicó que no existe un protocolo estándar para la teleoperación de robots CNC, por lo que fué necesario construirlo ex profeso para este proyecto.

5.2.2.3.1 Construcción del Cliente y Servidor.

La construcción del servidor sigue la estrategia de servidor con hilos (Figura 51) para la implementación del esquema cliente-servidor [30]. Para la atención de las solicitudes se crea un hilo de ejecución por cada cliente.

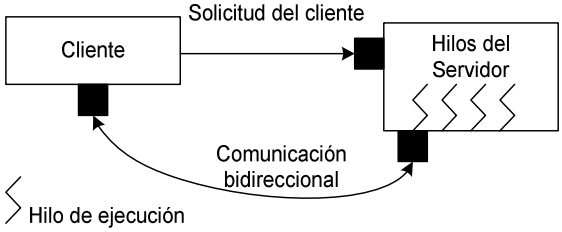


Figura 51 Implementación del servidor.

5.2.2.3.2 Protocolo de comunicación.

El objetivo del protocolo de comunicación es permitir el envío de comandos G, desde un equipo remoto para realizar la teleoperación del robot. Este protocolo se ubica en la capa de aplicación del modelo OSI y se implementa sobre los protocolos TCP/IP.

Modelo de operación

El protocolo está diseñado siguiendo el siguiente modelo de comunicación: El cliente establece una conexión con el servidor como resultado de una invocación

directa del usuario de la aplicación cliente. La aplicaciones cliente y servidor, están construidas con el *Framework*, las clases que se mencionan en la Figura 52 pertenecen al módulo de teleoperación. El envío de *comandos de solicitud* es generado por el *cliente* para pedir una acción en el servidor Los *comandos de respuesta* son generadas por el *servidor* como consecuencia de un comando de solicitud. La comunicación termina en los siguientes escenarios: el cliente termina la conexión, se interrumpe físicamente la comunicación y cuando se detiene la ejecución del servidor.

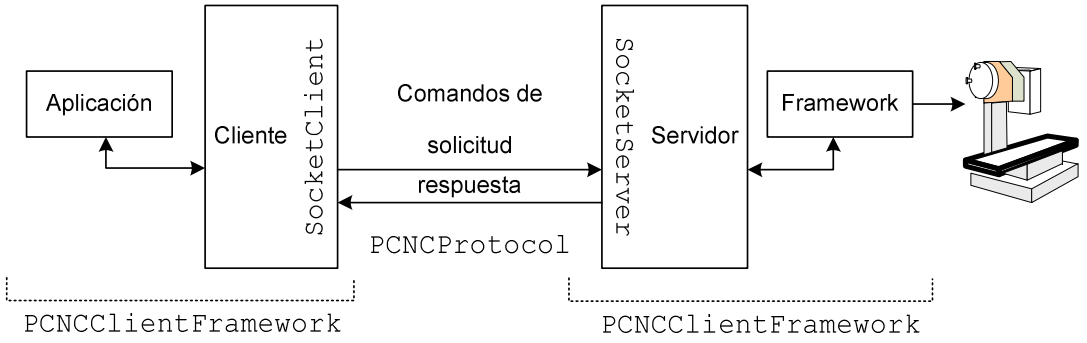


Figura 52 Modelo de operación del protocolo de comunicación

Instrucciones del protocolo de comunicación

El protocolo de comunicación se diseñó a partir de los casos de uso analizados en la sección 4.4.2.3. En la Figura 53 se muestra un diagrama de secuencia de la forma de operación de estas instrucciones y la invocación de los métodos del Framework.

Instrucciones para el caso de uso Ingresar

Es el primer comando que debe enviarse al servidor para iniciar una sesión de trabajo en el servidor.

Tabla 16 Comando para ingresar al servidor

Tipo de instrucción		
Solicitud	Respuesta	Descripción
`INGRESAR <usuario>`		Solicita acceso para trabajar en el servidor. Recibe como parámetro el `<usuario>`.
	`ACEPTADO`	El usuario es aceptado para trabajar en el servidor y se mantiene la conexión.
	`DENEGADO`	El usuario ya se encontraba trabajando en el servidor es rechazada esta solicitud y terminada la conexión.

Instrucciones para el caso de uso Enviar

Estos comandos se usan para transferir la secuencia de comandos G, del cliente al servidor. Se ofrecen dos modos de envío que le dan al usuario flexibilidad en el transferencia de los comandos G.

Tabla 17 Comandos para el envío de programas RS274-D

Tipo de instrucción		
Solicitud	Respuesta	Descripción

ENVIAR_NUEVO <comandos g>		Se limpia el contenido del objeto input, administrado por el Framework y se asignan los nuevos comandos G, recibidos en el servidor.
	RECIBIDO	Una vez que han sido depositados los comandos en el objeto input, se notifica al usuario.
	ERROR_NUEVO	En caso de que se presente un error, se notifica al cliente.
ENVIAR_AGREGAR <comandos g>		Se agregan los comandos G recibidos, al objeto input del Framework.
	RECIBIDO	Una vez que han sido depositados los comandos en el objeto input, se notifica al cliente.
	ERROR_AGREGAR	En caso de que se presente un error, se notifica al cliente.

Instrucciones para el caso de uso Ejecutar

Está formado por un conjunto de instrucciones que permiten tener control sobre la operación del robot.

Tabla 18 Comandos para el control de la ejecución de un programa RS274-D

Tipo de instrucción		
Solicitud	Respuesta	Descripción
EJECUTAR		Solicita el procesamiento de los comandos que fueron enviados previamente.
	TERMINADO	Al terminar la ejecución se notifica al cliente
	ERROR_EJECUTAR	En caso de que se presente un error, se notifica al cliente.
AVANCE		Solicita el grado de avance que se tiene sobre la ejecución de los comandos
	AVANCE <porcentaje>	Devuelve el grado de avance sobre el trabajo actual, este valor se devuelve en porcentaje.
CANCELAR		Interrumpe la ejecución de comandos.
	CANCELADO	Se notifica al cliente.

Instrucciones para el caso de uso Salir

Tabla 19 Comandos para terminar la sesión de trabajo

Tipo de instrucción		
Solicitud	Respuesta	Descripción
SALIR		Solicita la terminación de la conexión entre el cliente y servidor.
	ADIOS	Confirma el cierre de la conexión.

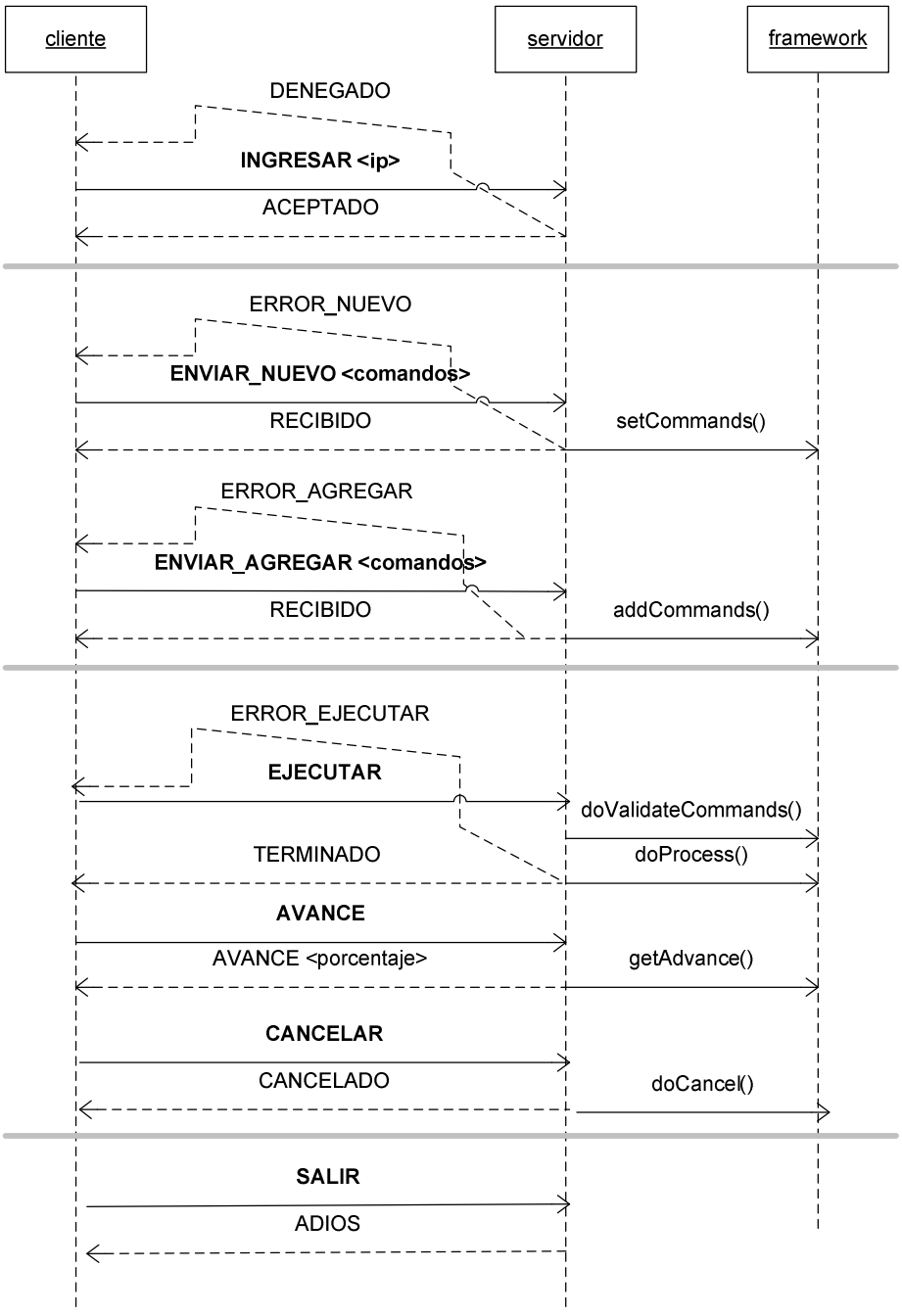


Figura 53 Diagrama de secuencia del protocolo de teleoperación de robots CNC

5.2.3 Caso de estudio: Control del robot MAXNC-10

El objetivo específico 4, indica desarrollar un controlador para el robot MAXNC-10 a partir del Framework. Se ha seleccionado el robot MAXNC-10 para probar las bondades del Framework, porque se tiene acceso a él, en el Grupo de Robótica y Análisis de Imágenes (Figura 54). El trabajo consiste en desarrollar el driver para controlar el robot por medio del puerto paralelo.

Figura 54 Robot MAXNC-10

5.2.3.1 Construcción del driver

De acuerdo a la vista lógica de la arquitectura que se muestra en la sección 4.5.1.2, para controlar a un nuevo tipo de robot, se debe escribir una clase que herede de `GAbstractDriver` y especialice algunos de sus métodos virtuales. El método `doProcess` de la clase `GAbstractDriver`, está diseñado cumpliendo con el *patrón de diseño plantilla*, esto significa que implementa la lógica para procesar las instrucciones RS274-D, pero delega en las clases derivadas la implementación de los métodos encargados del control de bajo nivel de los motores a pasos.

Para el robot MAXNC-10, se escribió la clase `GMaxNCDriver`, esta redefine el método `moveXYZ()` (Figura 55). Se diseñó como una *fábrica abstracta*, que es responsable de crear una instancia de la clase Max10.

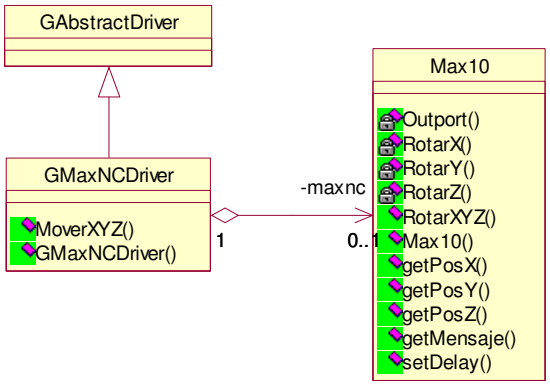


Figura 55 Implementación del controlador MaxNCDriver

Es en la implementación del método `moveXYZ()` donde se deben considerar aspectos inherentes al control de los motores y tomar en cuenta la dinámica como la inercia para controlar el desplazamiento o la compensación por backlash. Estos conceptos son independientes del Framework, dependen completamente de la configuración física del robot.

En la construcción del driver para el robot MAXNC-10, no se ha considerado la inercia debido a que trabaja con velocidades muy bajas.

5.2.3.1.1 Detalles técnicos del robot MAXNC-10

Desplazamiento máximo en el eje X	8”
Desplazamiento máximo en el eje Y	6”
Desplazamiento máximo en el eje Z	4”
Tamaño de la mesa de trabajo	11.9” largo x 4” de ancho
Velocidad máxima del motor	300 RPM
Ángulo de rotación por paso	1.8º
Resolución	0,00025”

5.2.3.1.2 Implementación del driver para el robot MAXNC-10

El constructor recibe como parámetro la dirección del puerto paralelo, que usará para conectarse físicamente al robot y redefine el valor de la constante H_RESOLUCION. Para determinar el valor de esta constante se realiza el siguiente cálculo.

Dadas las siguientes variables

$\alpha = 1.8^\circ$ Ángulo de rotación por paso.

$\delta = 0.05''$ Distancia en pulgadas entre cuerdas del tornillo.

Se tiene

$np = 360^\circ/\alpha = 200$, Número de pasos para completar una revolución.

$nr = 1''/\delta = 20$, Número de revoluciones para completar una pulgada.

H_RESOLUCION = $np*nr$, Número de pasos para realizar un movimiento equivalente a una pulgada.

H_RESOLUCION=200*20 = **4000**

La resolución que se puede obtener con este robot es la siguiente

r = 1” / H_RESOLUCIÓN = 0.00025”

```
GMaxNCDriver::GMaxNCDriver(int base)
{    maxnc = new Max10(base);
     H_RESOLUCION= 4000;   //-- (360grad/1.8 grad)*(1”/0.05”)
}
bool GMaxNCDriver::moveXYZ(double x, double y, double z)
{        return maxnc->RotarXYZ(x,y,z);
}
```

5.2.3.1.3 Configuración del puerto paralelo para el robot MAXNC-10.

De acuerdo a las pruebas que se realizaron, se seleccionó una secuencia completa, activando dos bobinas de cada motor para completar el ciclo.

La configuración del puerto paralelo para el robot se muestra en la Figura 56.

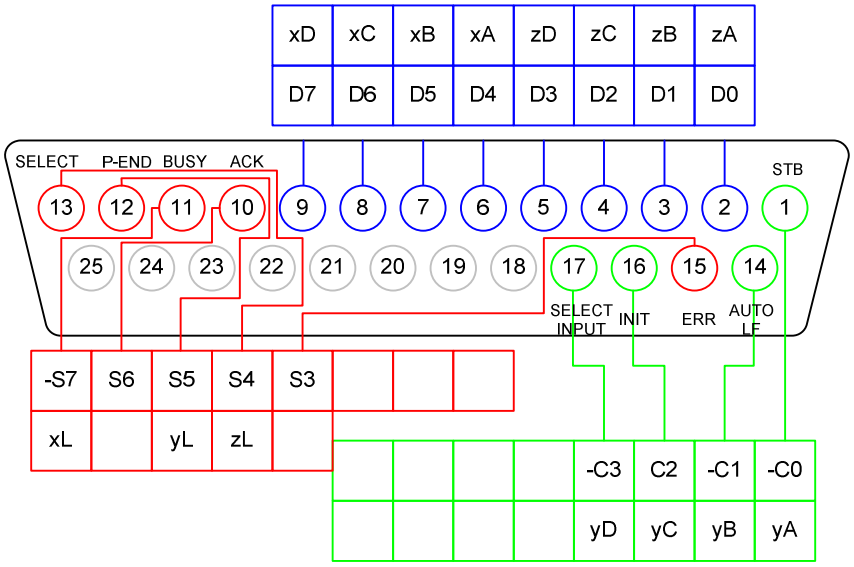


Figura 56 Configuración del puerto paralelo para el robot MAXNC-10

Los pasos que se deben enviar al puerto paralelo para una configuración de paso completo se muestra a continuación. Por la forma en que está acomodado el motor en el eje X, se debe seguir un orden inverso de la secuencia de pasos mostrada en la Tabla 11, para realizar los movimientos correctamente.

Tabla 20 Control del motor del eje X. Registro de datos

7	6	5	4	3	2	1	0			On
			XA					D0	XA	1
		XB						D1	XB	1
	XC							D2	XC	1
XD								D3	XD	1
Secuencia de pasos										
1	1	0	0	0	0	0	0	192		
1	0	0	1	0	0	0	0	144		
0	0	1	1	0	0	0	0	48		
0	1	1	0	0	0	0	0	96		

El movimiento del motor **Y** se realiza a través registro de control, para construir la secuencia de pasos se debe tomar en cuenta que el signo de los bits 0,1 y 3 se interpretan de forma invertida de acuerdo a la Tabla 10.

Tabla 21 Control del motor del eje Y. Registro de control

7	6	5	4	3	2	1	0			On
							YA	Strobe	YA	0
						YB		AutoF	YB	0

					YC			Init	YC	1
				YD				Selec	YD	0
Secuencia de pasos										
				-0	1	-1	-1	7		
				-1	1	-0	-1	13		
				-1	0	-0	-0	8		
				-0	0	-1	-0	2		

El movimiento del motor Z se realiza a través de los primeros 4 bits del registro de datos, la secuencia se construye como se explicó en la Tabla 10.

Tabla 22 Control del motor del eje Z. Registro de datos

7	6	5	4	3	2	1	0			On
							ZA	D0	ZA	1
						ZB		D1	ZB	1
					ZC			D2	ZC	1
				ZD				D3	ZD	1
Secuencia de pasos										
				1	1	0	0	12		
				0	1	1	0	6		
				0	0	1	1	3		
				1	0	0	1	9		

5.2.3.1.4 Implementación del controlador de los motores a pasos

```
int  Max10::RotarX(int direccion)
{       if ( ! direccion ) return 0;
        pos_secuencia_x+=direccion;
        if ( pos_secuencia_x < 0 ) pos_secuencia_x =MAX_SECUENCIAS-1;
        if ( pos_secuencia_x >= MAX_SECUENCIAS ) pos_secuencia_x =0;
        position_x += (direccion*H_INCREMENTO) ;
        last_mov.datos   =last_mov.datos & (~bits_x.datos  );
        last_mov.datos   =last_mov.datos | secuencia_x[pos_secuencia_x];
        return 1;
}
int  Max10::RotarXYZ(int direccionX,int direccionY, int direccionZ )
{       RotarX(direccionX);
        RotarY(direccionY);
        RotarZ(direccionZ);
        Outport(last_mov);
        return 1;
}
void Max10::Outport(LPT mensaje)
{       Out32(base  ,mensaje.datos);
        Out32(base+2,mensaje.control);
        delay();
}
```

5.2.4 Caso de estudio: Control del robot virtual en OpenGL

El objetivo específico 5, indica desarrollar una aplicación de escritorio que permita la edición, simulación y ejecución de comandos RS274-D, construida usando el Framework. El desarrollo de este objetivo se divide en dos etapas, primero se construye un controlador para un robot virtual en OpenGL, que servirá para la simulación de los comandos G en pantalla. Posteriormente se construirá la aplicación de escritorio usando el driver de este robot virtual.

5.2.4.1 Construcción del driver

De acuerdo a la vista lógica de la arquitectura, para permitir el control de nuevos tipos de robots, se debe construir una clase derivada de *GAbstractDriver*. Para la construcción del controlador en OpenGL (Figura 57), se deben implementar todas las rutinas usadas por el *método plantilla* `doProcess()`.

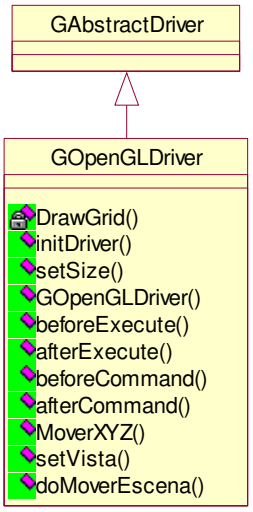


Figura 57 Implementación del controlador GOpenGLDriver

5.2.4.1.1 Constructor: GOpenGLDriver()

El constructor recibe como parámetros el ancho y el alto del área de trabajo en pantalla, para que la biblioteca OpenGL realice las proyecciones correctamente. Debe redefinir el valor de la constante `H_RESOLUCION`, sin embargo como este no es un robot que exista físicamente, no depende del ángulo de rotación ni desplazamiento obtenido en cada paso.

El valor de `H_RESOLUCION` se asignará de acuerdo a la resolución que se desee obtener en pantalla, su significado en OpenGL será número de puntos por pulgada en pantalla. Los algoritmos de línea y arco, únicamente usan la instrucción OpenGL que sirve para dibujar píxeles en el espacio tridimensional.

Por ejemplo si se asigna un valor para `H_RESOLUCION = 100`; en un programa de G-Codes que solicite un movimiento lineal de una pulgada, en pantalla se va a trazar como una secuencia lineal de 10 puntos. Es importante notar que este valor tiene dos efectos, un número menor genera imágenes de baja resolución, un número mayor provoca que la simulación en pantalla se realice con lentitud.

5.2.4.1.2 Inicializar el controlador: initDriver()

Este método es invocado por el Framework y tiene el propósito de especificar el funcionamiento inicial del controlador. Para OpenGL, limpia el área de dibujo, especifica la perspectiva y coloca una cámara en proyección inicial.

5.2.4.1.3 Preparar la ejecución de los comandos con: beforeExecute() y afterExecute()

A diferencia de un robot real que ejecuta el programa solo una vez, en OpenGL se recorre todo el programa de G-Codes, cada vez que se necesita redibujar el escenario 3D, esto ocurre cuando se mueve la pantalla o se modifica su tamaño. El método `beforeExecute()` es invocado desde la clase `GabstractDriver` con el método `doProcess()` antes de iniciar la ejecución del primer comando. El método `afterExecute()` es invocado después de la ejecución del último comando del programa de G-Codes.

Estos métodos inicializan la cámara y las herramientas de dibujo que se usarán en la ejecución del programa.

5.2.4.1.4 Ejecución de comandos: beforeCommand() y afterCommand()

Estos métodos son invocados por el método `doProcess()`, antes y después de la ejecución de cada comando G-Code. La implementación realizada en OpenGL identifica el número de comando y cambia el color de los pixeles. A continuación se construyen los vértices con el método `moveXYZ()`, los vértices serán usados por OpenGL para dibujar los objetos en pantalla.

5.2.4.1.5 Implementación del driver: moveXYZ()

El método que realiza el movimiento físico de los motores es `moveXYZ()`. La implementación en OpenGL, no usa una secuencia de pasos de pasos, porque no tiene un motor que mover, en su lugar modifica la coordenada `move`, que guarda la posición a donde se moverá el robot, incrementando o decrementado sus valores actuales. Esta coordenada se usa para generar un vértice por cada paso generado desde los algoritmos de línea o arco.

Los parámetros recibidos x,y,z, pueden tener los valores +1,0,-1, que especifican el sentido de movimiento del paso, este valor se divide entre H_RESOLUCION para ajustar el valor de paso a las coordenadas usadas en OpenGL.

```
bool GOpenGLDriver::moveXYZ(double x, double y, double z)
{    move.x+= x / H_RESOLUCION ;
     move.y+= y / H_RESOLUCION ;
     move.z+= z / H_RESOLUCION ;
     glVertex3d(move.x,move.y,move.z);
     return false;
}
```

5.2.4.1.6 Métodos complementarios: setVista() y doMoverEscena()

El controlador para OpenGL declara dos métodos que no se encuentran en la clase `GAbstractDriver`, pero que son necesarios para realizar operaciones adicionales `setVista()` y `doMoverEscena()`. El primero permite visualizar la escena desde diferentes perspectivas: frente, arriba, derecha e isométrico. El segundo permite mover libremente el escenario, rotación y traslación en los ejes X,Y,Z. Ambos métodos deben ser invocados desde una aplicación que use el Framework.

5.2.5 Caso de estudio: Aplicación IDE-CNC

En esta sección se concluye el desarrollo del objetivo específico 5. Se usan los controladores desarrollados anteriormente, el robot virtual se usa para ofrecer la funcionalidad de simulación y el controlador MAXNC-10 para trabajar con un robot real. La aplicación usa el Framework para tener acceso a las funciones de control y teleoperación.

En la Figura 58 se muestran los componentes de interfaz gráfica, el código detrás de estos componentes, hace uso del Framework para obtener su funcionalidad, quien a su vez solicita los servicios proporcionados por las clases de los módulos de Control CNC y Teleoperación.

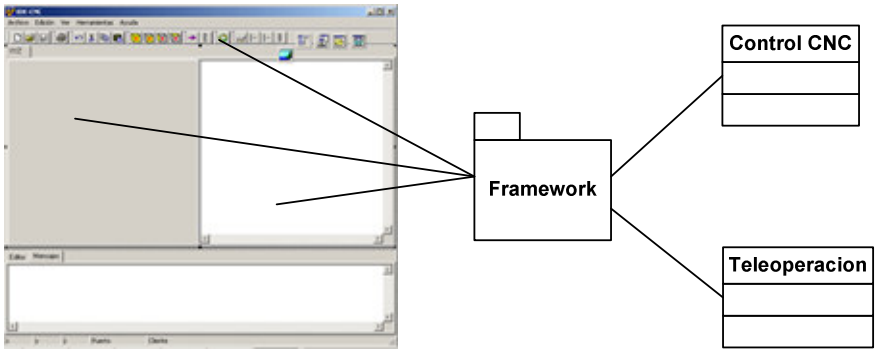


Figura 58 Integración de la interfaz gráfica y el Framework

5.2.5.1 Diseño de clases.

La implementación del Framework se realiza por composición, la aplicación funciona como un patrón *fabrica abstracta* para los controladores de cada robot. En la Figura 59, se puede ver la relación entre la aplicación y el Framework.

5.2.5.2 Constructor TIDEEditor

En el constructor de la aplicación se crean dos objetos de la clase GOpenGLDriver y GMaxNCDriver, que servirán para controlar el robot virtual (controlOpenGL) y el MAXNC-10 (controlMaxNC) respectivamente, estos objetos son usados en el constructor por el Framework, para que el envío de mensajes (Figura 60).

A continuación se crean dos objetos de la clase `GFramework`, asociando a cada objeto el controlador del robot respectivo, para OpenGL se crea el objeto fwOpenGL y para el robot MAXNC-10 se crea el objeto fwMaxNC.

También se leen los valores iniciales de la aplicación desde un archivo de configuración, en caso de que no exista se usarán valores por defecto.

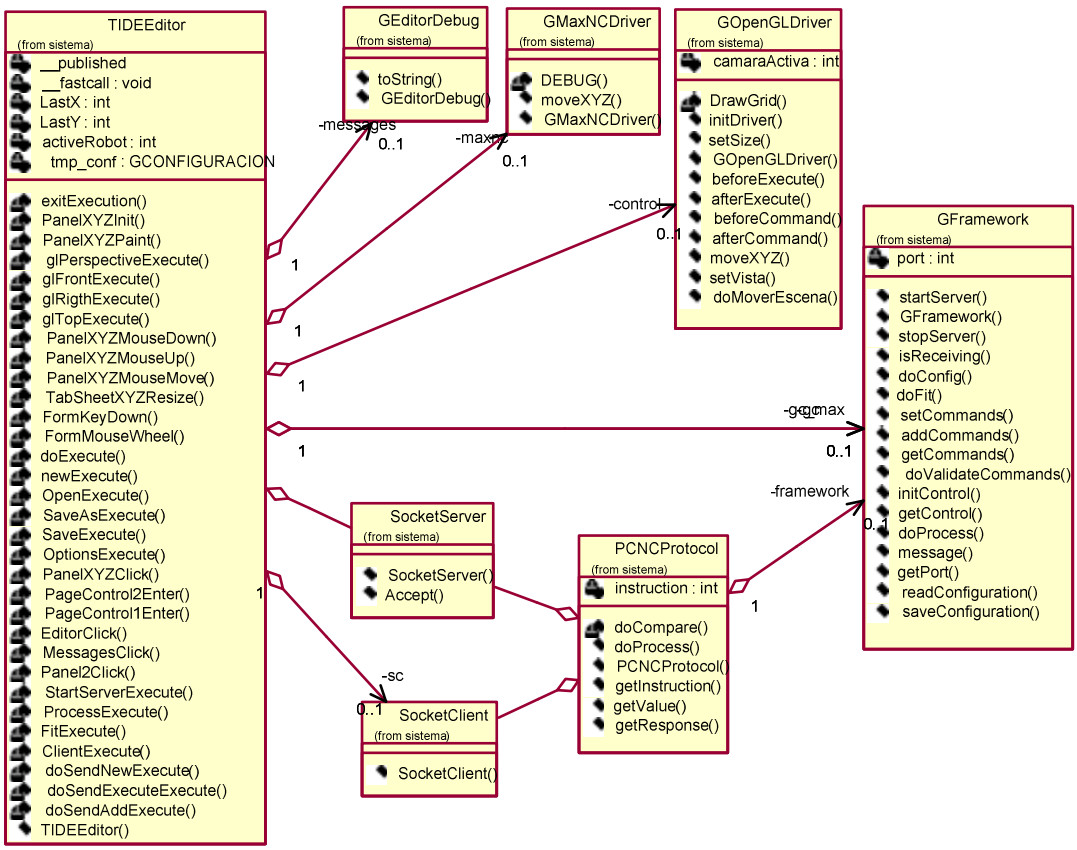


Figura 59 Diagrama de clases de la aplicación IDE-CNC

```
__fastcall TIDEEditor::TIDEEditor(TComponent* Owner)
        : TForm(Owner)
{ //-- framework para leer parámetros
  ... leer archivo de configuracion
//-- Objeto para enviar mensajes de error
messages= new GEditorDebug(Mensajes);
messages->setLevel(tmp_conf.leveldebug);
//-- Construir el driver y Framework - OpenGL
controlOpenGL = new GOpenGLDriver(PanelXYZ->Width,PanelXYZ->Height );

fwOpenGL = new GFramework(messages, controlOpenGL,  tmp_conf.serverport);
//-- Construir el driver y Framework - MaxNC10
controlMaxNC = new GMaxNCDriver(tmp_conf.parallelport);

fwMaxNC = new GFramework(messages, controlMaxNC, tmp_conf.serverport);
//-- Comportamiento iniciar de la aplicacion
serverIsRunning = false;
clientIsConnected  = false;
}
```

Figura 60 Constructor de la aplicación IDE-CNC

5.2.5.3 Descripción general.

La aplicación ofrece todas las funciones definidas en los casos de uso. Editor, Simulador, Teleoperación y control, opera en los modos cliente y servidor. La Figura 61 muestra la distribución de los componentes en la aplicación.

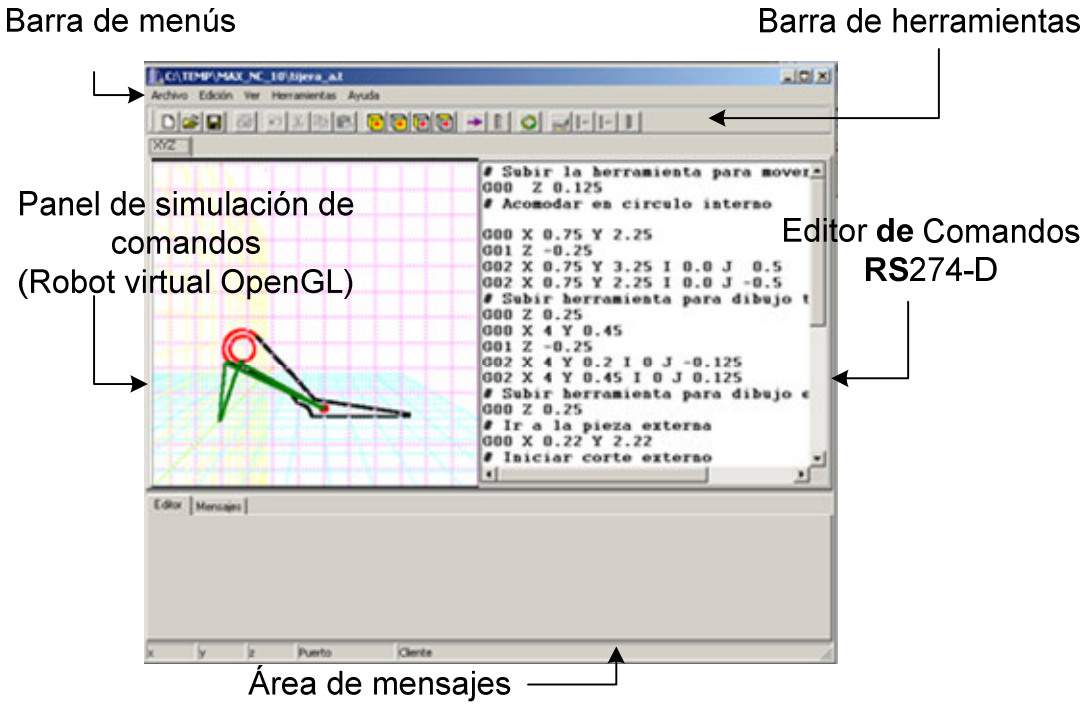


Figura 61 Elementos principales de la aplicación IDE-CNC

5.2.5.3.1 Simulador de comandos

El simulador de comandos usa el controlador del robot virtual OpenGL para crear una versión tridimensional de programa RS274-D, se compone del panel de simulación (Figura 62), las funciones para visualizar desde diferentes perspectivas y el comando que solicita la simulación del programa (Figura 67).

Panel de simulación

Es el espacio de la aplicación que se usa para mostrar la simulación de los programas RS274-D, permite realizar movimientos de rotación y traslación en los tres ejes, por medio del teclado o ratón (Figura 62).

En CBuilder existe un componente para visualizar los gráficos generados con las funciones OpenGL, en su evento de inicialización, se especifica el espacio disponible para dibujar, estos valores son enviados al controlador del robot virtual para realizar las proyecciones necesarias. El robot virtual se inicializa a través de la instancia del Framework en el objeto `fwOpenGL` invocando el método `initControl()`. Se invoca el método `doProcess()` del Framework para que regenere la simulación del programa en caso de mover o cambiar el tamaño de la ventana.

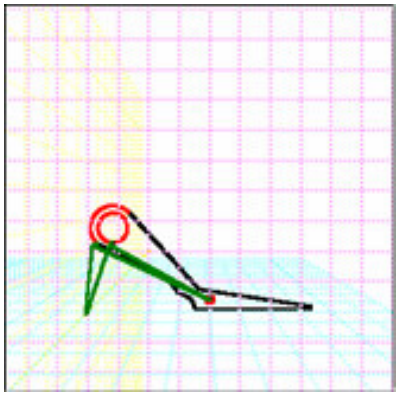


Figura 62 Panel de simulación

```
void __fastcall TIDEEditor::PanelXYZInit(TObject *Sender)
{//-- Especificar el espacio disponible para dibujar
 PanelXYZ->Width = ( TabSheetXYZ->Width < TabSheetXYZ->Height
                        ? TabSheetXYZ->Width
                        : TabSheetXYZ->Height );
 PanelXYZ->Height =PanelXYZ->Width;
 //-- Indicar al controlador OpenGL, las dimensiones de la ventana
   GOpenGLDriver *go_aux =(GOpenGLDriver *)fwOpenGL->getControl();
                 go_aux->setSize(PanelXYZ->Width , PanelXYZ->Height);
   fwOpenGL->initControl();
   fwOpenGL->doProcess();
}
```

Visualización desde diferentes perspectivas

Modifica el punto de visión del escenario desde 4 posiciones: Perspectiva, frente, lateral y arriba, Figura 63 a Figura 66. Estas funciones existen únicamente para el driver de OpenGL, no pertenecen a la definición de la clase base `GAbstractDriver`. Para que la aplicación pueda usar estos métodos se le pide al Framework una copia de la instancia del driver.

```
void __fastcall TIDEEditor::glPerspectiveExecute(TObject *Sender)
{
        ((GOpenGLDriver *)fwOpenGL->getControl())->setVista(1);
         PanelXYZ->Repaint();
}
```

Estas funciones pueden ser solicitadas desde la barra de menús y la barra de herramientas.

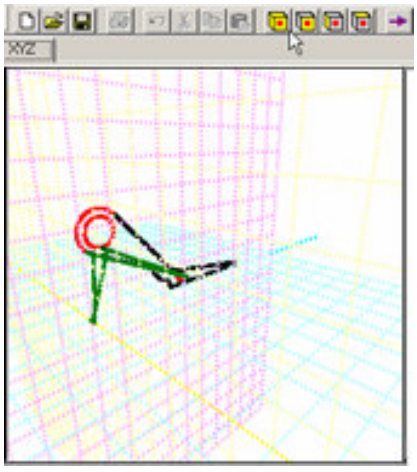


Figura 63 Vista en perspectiva

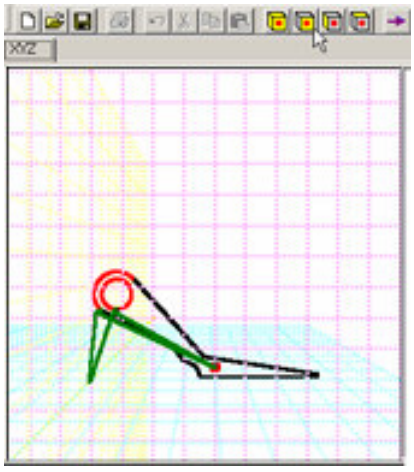


Figura 64 Vista de frente

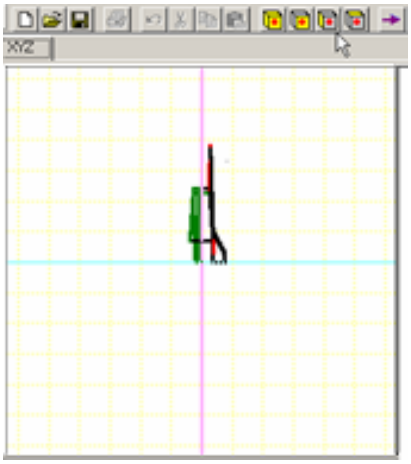


Figura 65 Vista lateral

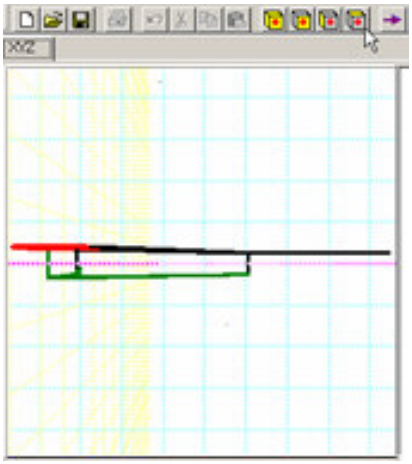


Figura 66 Vista desde arriba

Simulación de comandos RS274-D

La simulación de los comandos es generada con el driver del robot virtual. Se debe cargar o escribir un programa en el editor de comandos y solicitar la simulación (Figura 67).

Figura 67 Icono para solicitar la simulación de comandos

El botón de simulación realiza los siguientes pasos:

1. Lee el contenido del editor de comandos.
2. Envía este contenido al Framework con el método `setComandos()`.
3. Solicita la validación de los comandos al Framework.
4. Redibuja el panel de simulación.

```
void __fastcall TIDEEditor::ExecuteExecute(TObject *Sender)
{       Mensajes->Lines->Clear();
        fwOpenGL->setCommands(Editor->Lines->GetText());
        fwOpenGL->doValidateCommands();
        PanelXYZInit(Sender);
        PanelXYZ->Repaint();
}
```

Procesamiento de comandos en el robot MAXNC-10

El procesamiento de los comandos en el robot MAXNC-10 se realiza con el driver `GMaxNCDriver`. Para realizar un pieza se debe cargar o escribir un programa en el editor de comandos y solicitar el procesamiento (Figura 68).

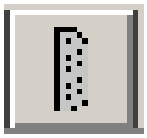


Figura 68 Icono para solicitar el procesamiento en el robot MAXNC-10

El botón de *procesar* realiza los mismos pasos que para la simulación, pero usando el Framework asociado al robot MAXNC-10.

```
void __fastcall TIDEEditor::ProcessExecute(TObject *Sender)
{ //-- Usar el framework asociado al robot MaxNC10
         fwMaxNC->setCommands(Editor->Lines->GetText());
         fwMaxNC->doValidateCommands();
         fwMaxNC->initControl();
         fwMaxNC->doProcess();
}
```

5.2.5.3.2 Editor de comandos

El editor de comandos se implementa usando el componente TMemo provisto por CBuilder, incluye las funciones para crear un archivo nuevo, abrir, guardar y guardar como. La invocación de estas funciones se puede realizar desde la barra de menús o la barra de herramientas (Figura 69). El editor incluye funciones propias de las aplicaciones de texto: Seleccionar todo, Cortar, Copiar y Pegar (Figura 70).

Figura 69 Funciones del editor de comandos

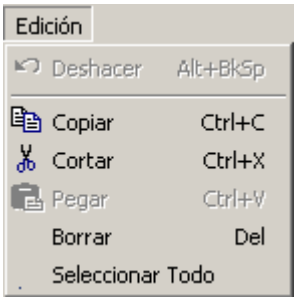


Figura 70 Funciones de edición

En este módulo no se usan métodos proporcionados por el Framework.

5.2.5.4 Servidor de Teleoperación de robots CNC

El Framework proporciona las funciones para iniciar y detener el servidor de comandos remotos. El componente de teleoperación se encarga de recibir las instrucciones por medio del protocolo y transferirlas al Framework

La aplicación IDE-CNC selecciona el objeto framework del robot activo. Si el servidor de este robot no está recibiendo peticiones, se inicia con el método `startServer()`,en caso contrario el servidor es detenido con el método `stopServer()`.

```
void __fastcall TIDEEditor::DoStartServerExecute(TObject *Sender)
{  if (serverIsRunning)
   {  //-- Mostrar el port del servidor
      ...
      switch (robotactivo)
      {       case 1: fwOpenGL->stopServer(); break;
              case 2: fwMaxNC->stopServer(); break;
      }
   }
   else
   {   switch (robotactivo)
        {     case 1:
                  . . .
               fwOpenGL->startServer();
              break;
                case 2:resolucion = 2000; break;
                  . . .
               fwMaxNC->startServer();
        }
   }
}
```

El servidor puede ser iniciado o detenido con los íconos que se muestran en la Figura 71, estos iconos se encuentran en la barra de herramientas o en la opción herramientas del menú principal.


Iniciar servidor


Detener servidor

Figura 71 Iniciar/detener el servidor

5.2.5.5 Cliente para la teleoperación de robots CNC

El modo cliente se obtiene activando la opción Conectar, desde el menú herramientas o directamente en la barra de herramientas. Su implementación se realiza usando los métodos proporcionados por el Framework con el objeto `fwOpenGL`, para los comandos del protocolo de comunicación.

Iniciar servidor

```
void __fastcall TIDEEditor::DoConnectClientExecute(TObject *Sender)
{  if ( clientIsConnected )
```

```
    {   clientIsConnected = fwOpenGL->disconnect();
        ...
    }
    else
    {   clientIsConnected = fwOpenGL->connect(tmp_conf.cliente_ip,
                                              tmp_conf.cliente_port);
        if (clientIsConnected)
        { fwOpenGL->doSendEnter(tmp_conf.cliente_ip);
          ...
        }
        else
        {  Application->MessageBox(
           "No se pudo establecer conexión con el servidor, intentar
            despues.",NULL, MB_OK    );
        }
    }
}
```

Enviar comandos al servidor

```
void __fastcall TIDEEditor::DoSendNewExecute(TObject *Sender)
{    if (clientIsConnected)
     fwOpenGL->doSendSetCommand(Editor->Lines->GetText());
}
```

Solicitar la ejecución en el servidor

```
void __fastcall TIDEEditor::DoSendExecuteExecute(TObject *Sender)
{       if (clientIsConnected)
        fwOpenGL->doSendExecute();
}
```

5.3 Pruebas

Las pruebas son el proceso de encontrar diferencias entre el comportamiento esperado que se especificó en los casos de uso y el comportamiento observado en el sistema [5].

5.3.1 Condiciones para el desarrollo de los experimentos

Para verificar el funcionamiento del Framework, se diseñaron dos piezas para la fabricación de una tijera (Figura 72 y Figura 73). El programa se escribió usando unidades de medida en pulgadas. La resolución del robot es de 0.00025”. La herramienta de corte tiene una dimensión de 0.0625”, el comando de compensación de herramienta no se usó porque no está implementado en el driver abstracto. La aplicación servidor se instaló en una computadora Pentium III, con sistema operativo Windows 98, en este mismo equipo se conecto el robot MAXNC-10 en el puerto paralelo LPT1. El programa cliente se instaló en un equipo Pentium IV. La comunicación entre el cliente y servidor se realizó por medio del protocolo TCP/IP en una red de área local. La dirección IP del servidor es 148.204.20.37 recibiendo peticiones en el puerto 2000, y el cliente tiene la dirección IP 148.204.20.82.

5.3.1.1 Descripción de la pieza A para fabricar una tijera

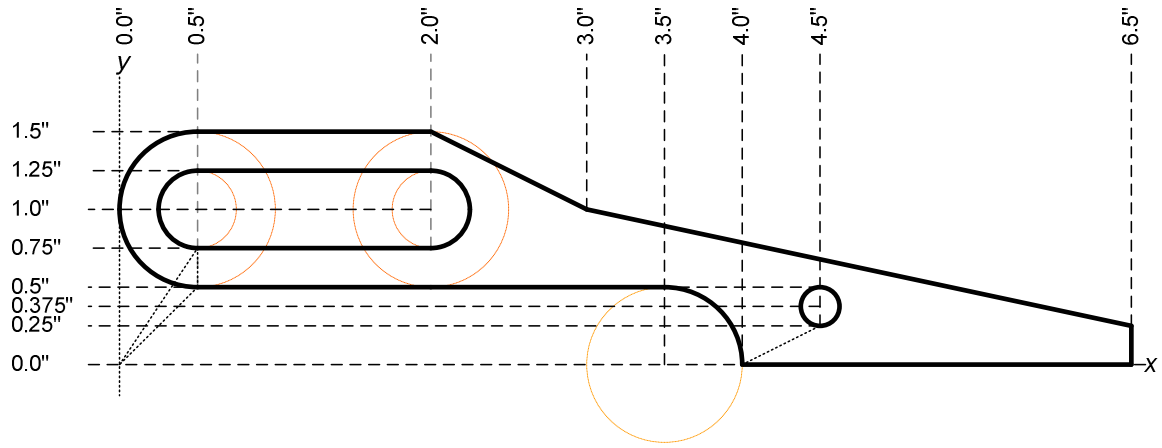


Figura 72 Tijera pieza A

Programa RS274-D para la fabricación de la pieza A.

```
#Tijera Pieza A
G20                 # Unidades en pulgadas
G90                 # Modo de coordenadas absolutas
G00 Z 0.125         # Subir herramienta sin corte
G01 X0.5 Y 0.75     # Mover a la sección interna
G01 Z-0.25          # Bajar a corte interno
G02 X0.5 Y 1.125 I0 J0.25
G01 X2 Y1.25
G02 X2 Y0.75 I0 J-0.25
G01 X0.5 Y0.75
G01 Z0.25
G00 X0.5 Y0.5       # Cortar mango de la tijera
G01 Z-0.25
G02 X0.5 Y1.5 I0 J0.5
G01 X2 Y1.5
G01 X3 Y1
G01 X8 Y.25
G01 X8 Y0
G01 X4 Y0
G01 Z0.25
G00 X4.5 Y0.25      # Cortar orificio para tornillo
G01 Z-0.25
G02 X4.5 Y0.5 I0 J0.125
G02 X4.5 Y0.25 I0 J-0.125
G01 Z0.25
G00 X4 Y0           # Regresar a corte del mango
G01 Z-0.25
G03 X3.5 Y0.5 I-.5 J0
G01 X0.5 Y0.5       # Fin de toda la pieza
G01 Z0.25
G00 X0Y0            # Volver a coordenada HOME
G00 Z0
```

5.3.1.2 Descripción de la pieza B para fabricar una tijera

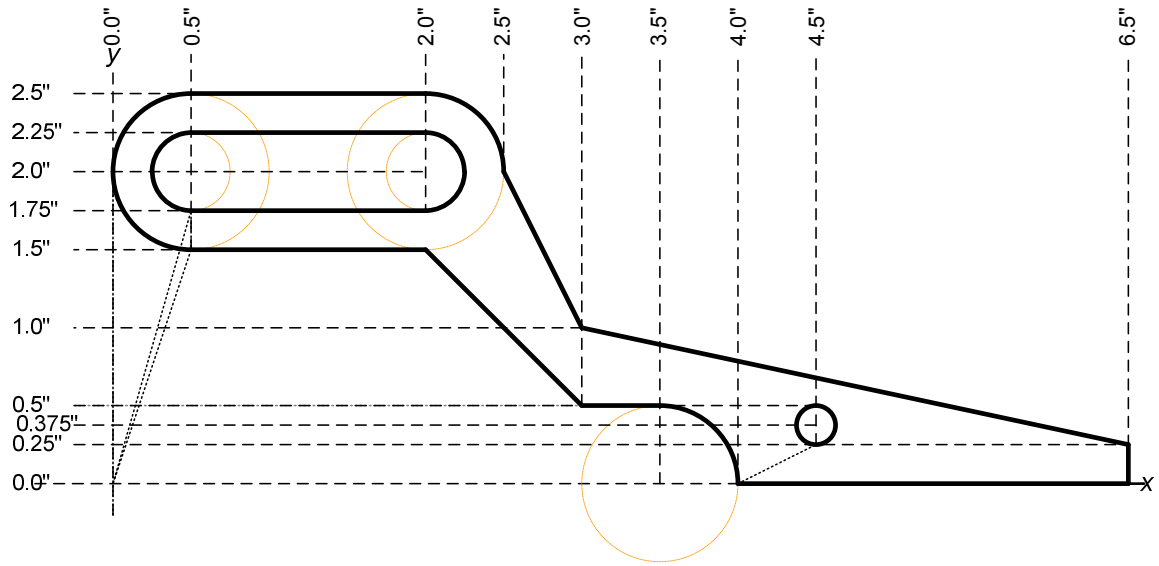


Figura 73 Tijera pieza B

Programa RS274-D para la fabricación de la pieza B.

```
#Tijera Pieza B
G20                 # Unidades en pulgadas
G90                 # Modo de coordenadas absolutas
G00 Z 0.125         # Subir herramienta sin corte
G00 X0.5 Y 1.75     # Mover a la sección interna
G01 Z-0.25
G02 X0.5 Y 2.125 I0 J0.25
G01 X2 Y2.25
G02 X2 Y1.75 I0 J-0.25
G01 X0.5 Y1.75      # Final de corte interno
G01 Z0.25
G00 X0.5 Y1.5       # Cortar mango de la tijera
G01 Z-0.25
G02 X0.5 Y2.5 I0 J0.5
G01 X2 Y2.5
G02 X2.5 Y2 I0 J-.5
G01 X3 Y1
G01 X6.5 Y.25       # Inicia punta de la tijera
G01 X6.5 Y0
G01 X4 Y0           # Fin de punta
G01 Z0.25
G00 X4.5 Y0.25      # Orificio tornillo
G01 Z-0.25
G02 X4.5 Y0.5 I0 J0.125
G02 X4.5 Y0.25 I0 J-0.125
G01 Z0.25
G00 X4 Y0           # Continuar corte de mango
G03 X3.5 Y0.5 I-.5 J0
G01 X3 Y0.5
G01 X2Y1.5
G01 X0.5Y1.5        # Fin de toda la pieza
G01 Z0.25
```

```
G00 X0Y0          # Volver a coordenada HOME
G00 Z0
```

5.3.2 Pruebas unitarias

Estas pruebas se realizan con el fin de determinar si un módulo de código funciona correctamente, para verificar la arquitectura del modelo propuesto se realizaron pruebas a los principales componentes de la arquitectura.

5.3.2.1 Módulo de Control CNC

Tiene como objetivo verificar que se interpretan y procesan correctamente los comandos en el estándar RS274-D. La prueba consiste en cargar el programa para realizar la pieza A en la aplicación de escritorio, solicitar que sea realice una simulación en pantalla y a continuación enviar el programa para que sea fabricado por el robot.

Para determinar que la prueba fué exitosa, se contrastarán las medidas de la pieza generada por el robot con el programa.

5.3.2.2 Módulo de Teleoperación

Tiene como objetivo verificar que es posible realizar la teleoperación de un robot CNC. La prueba consiste en cargar el programa para realizar la pieza B, desde la aplicación de escritorio en modo cliente, y solicitar que realice la ejecución en el servidor de forma remota.

Para determinar si la prueba fué exitosa, se debe mostrar el envío y recepción de comandos del protocolo con una herramienta de análisis de paquetes TCP.

5.3.3 Prueba de integración de clases

Esta prueba se realiza con el fin de verificar que al integrar clases y sus instancias en un producto de software pueden funcionar como un todo.

5.3.3.1 Uso del Framework

La prueba consiste en revisar que en la aplicación de escritorio IDE-CNC funcionan correctamente las porciones de código usadas para implementar los comportamientos de Control CNC y Teleoperación.

Esta prueba se califica como exitosa si las pruebas unitarias son exitosas.

5.3.4 Pruebas del sistema

Las pruebas del sistema encuentran diferencias entre los requerimientos planteados y el sistema desarrollado.

5.3.4.1 Seguimiento de requerimientos

La prueba consiste en dar seguimiento a los requerimientos partiendo desde la declaración de los objetivos específicos del proyecto de tesis, análisis, diseño implementación y las pruebas.

La prueba es exitosa si a cada objetivo específico se le puede dar seguimiento, presentando los resultados en una tabla que haga referencia a las secciones donde se realizó cada etapa.

5.4 Resultados

En las siguientes fotografías se muestran los equipos usados para la realización de los experimentos. La fotografía de la Figura 74, muestra la computadora usada para instalar la aplicación en modo servidor, este equipo tiene instalado el robot MAXNC-10 al puerto paralelo.

Figura 74 Robot MAXNC-10 Conectado al servidor

La fotografía de la Figura 75 muestra la computadora usada en modo cliente, en esta se instaló la aplicación para generar la pieza B de la tijera.

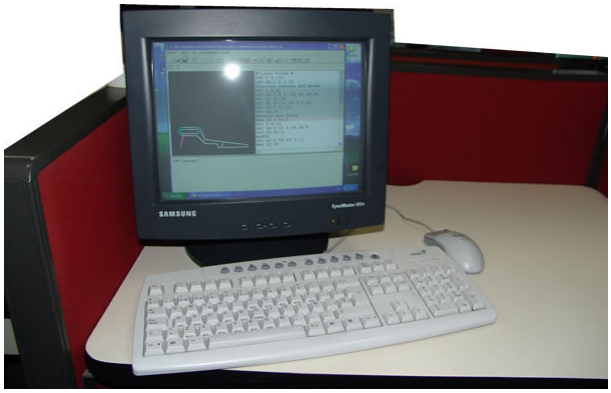


Figura 75 Cliente conectado al servidor

5.4.1 Resultado de las pruebas unitarias

La fotografía de la Figura 76 se tomo durante la realización de una de las piezas.

Figura 76 Robot MAXNC-10 fabricando las piezas de la tijera

5.4.1.1 Módulo de Control CNC

En la Figura 77 se muestra la pieza fabricada por el robot de forma local. El resultado de la prueba, permite concluir que, el módulo de control ofrecido por el Framework funciona correctamente.

- El intérprete reconoció los comandos RS274-D del programa.
- El driver abstracto implementa correctamente los algoritmos para procesar los comandos de línea y arco.
- El driver desarrollado para el robot MAXNC10, genera la secuencia correcta de pasos.
- El driver desarrollado para el robot virtual OpenGL, permite visualizar correctamente la simulación del programa en pantalla.

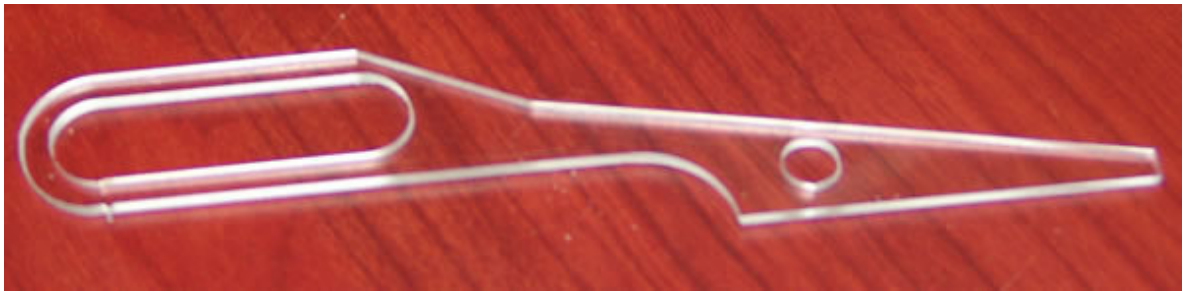


Figura 77 Fabricación de la pieza A de la tijera

Figura 78 Fabricación de la pieza B de la tijera

5.4.1.2 Módulo de Teleoperación

Para verificar la ejecución del protocolo de comunicación, se uso la herramienta de análisis de protocolos Ethereal. En la Figura 79 se muestran los paquetes capturados entre el cliente y servidor, en orden de izquierda a derecha, las columnas indican: número de paquete, dirección IP origen, dirección IP destino, protocolo de transporte y una breve muestra del encabezado del paquetes. En la parte inferior de la ventana se muestra una versión hexadecimal y decimal del contenido del paquete, en la figura se ha seleccionado el paquete número 7, que es el inicio de la comunicación entre el cliente y servidor, se puede ver el envío del comando ingresar.

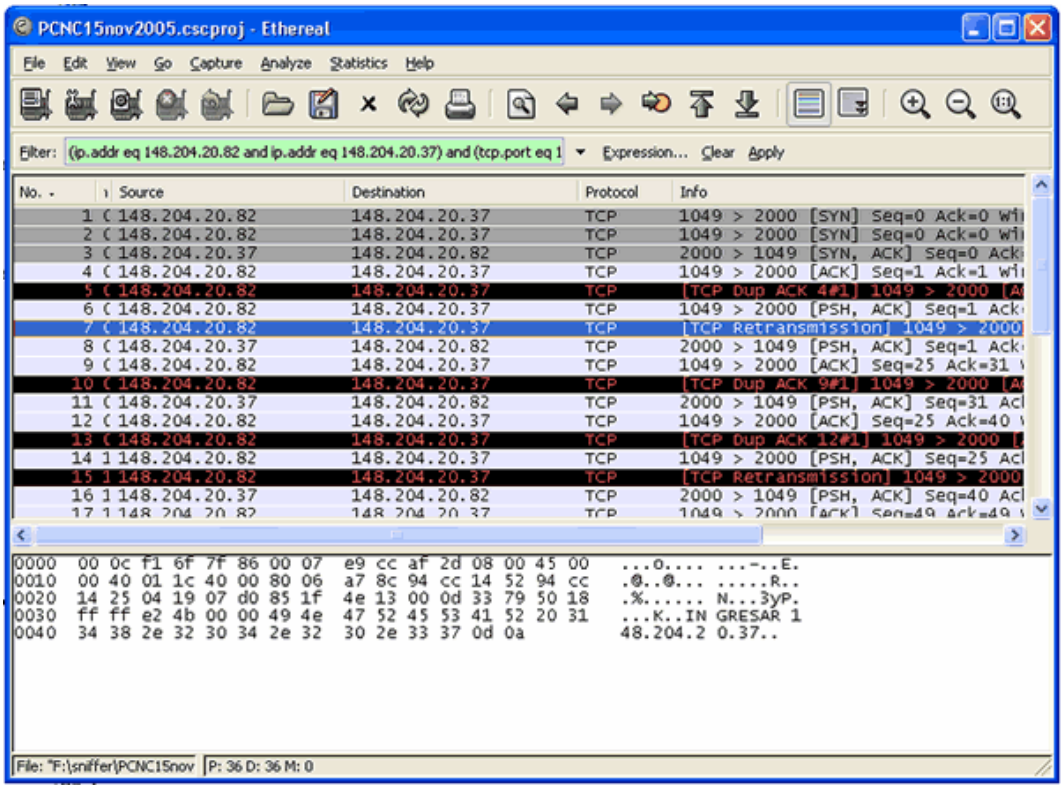


Figura 79 Captura de paquetes entre el cliente y servidor

Para visualizar la secuencia de comandos en forma legible, se usa la opción del software, para dar seguimiento a una secuencia de paquetes TCP en formato de texto, en la Figura 80 se puede apreciar el protocolo en pleno funcionamiento obtenido con el software Ethereal.

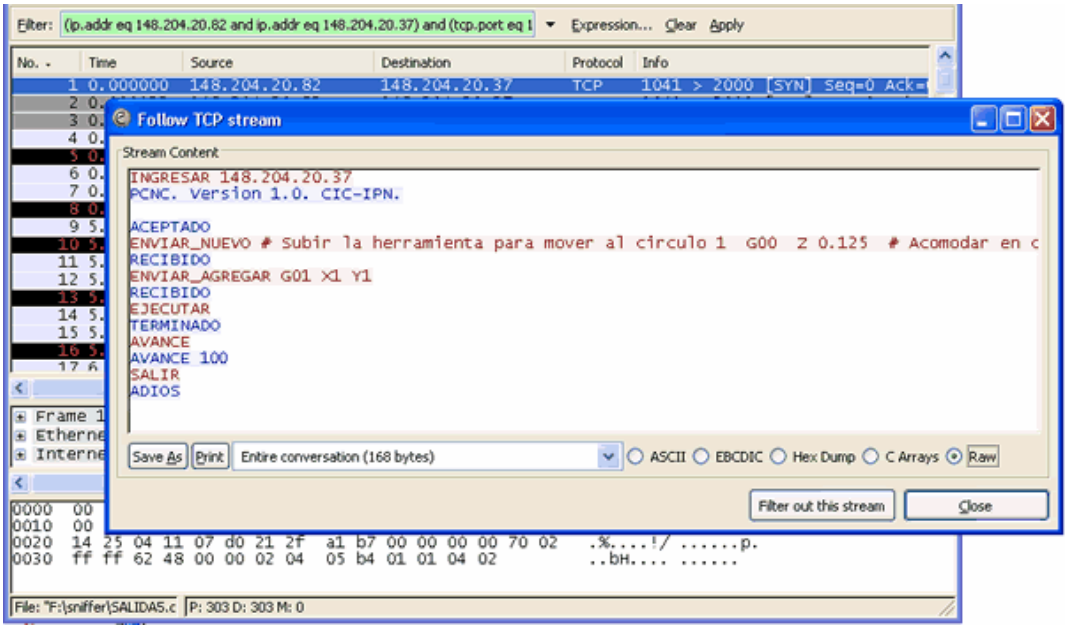


Figura 80 Seguimiento de paquetes TCP entre el equipo cliente y servidor

La pieza generada de forma remota se muestra en la Figura 78. El resultado de la prueba, permite concluir que, el módulo de teleoperación ofrecido funciona correctamente, ya que se pudo realizar la fabricación de la pieza B de forma remota. El funcionamiento se verificó con el software Ethereal.

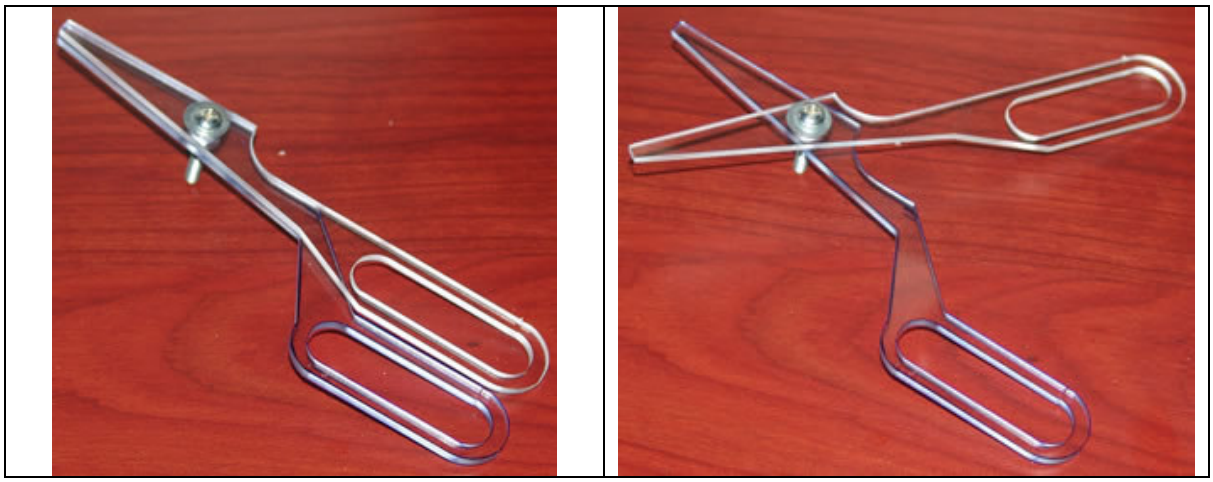


Figura 81 Tijera armada

5.4.2 Resultado de las pruebas de integración de clases

La prueba de integración se considera exitosa, porque para el desarrollo de la aplicación IDE-CNC se uso únicamente el Framework, esta actúa como una clase integradora do toda la arquitectura. Al ser exitosas las pruebas unitarias, se validaron que los componentes funcionan de acuerdo a lo esperado. La aplicación IDE-CNC es la evidencia de que las clases funcionan integradas como un todo.

5.4.3 Resultado de las pruebas de sistema

Seguimiento de requerimientos

En la Tabla 23, se puede observar que los objetivos específicos fueron cubiertos en las cuatro etapas de desarrollo.

El resultado de la prueba es exitoso, porque se muestra el seguimiento dado a cada uno de los objetivos específicos, los números debajo de cada etapa indican la sección del documento donde se realizó la actividad. Las etapas de análisis y diseño para construir el controlador del robot MAXNC-10 y la aplicación de escritorio, no se citan en el documento, porque son la aplicación de los objetivos específicos 1,2 y3.

Tabla 23 Seguimiento de requerimientos para las pruebas del sistema

Objetivo específico	Análisis	Diseño	Implementación	Prueba
Diseñar un Framework para el desarrollo de aplicaciones de control de robots de control numérico.	4.3 4.4.1.1	4.5.1.1	5.2.2	5.3.3 5.3.4
Diseñar un componente para el control de robots CNC, que interprete programas RS274-D.	4.4.1.2 4.4.2.1	4.5.1.2	5.2.2.2	5.3.2 5.3.2.1
Diseñar un protocolo de comunicación para la teleoperación de robots CNC.	4.4.1.3 4.4.2.3	4.5.1.3	5.2.2.3	5.3.2 5.3.2.2
Desarrollar un controlador para el robot MAXNC-10 a partir del Framework.			5.2.3	5.3.2 5.3.2.1
Desarrollar una aplicación de escritorio que permita la edición, simulación y ejecución de comandos RS274-D, construida usando el Framework.			5.2.3, 5.2.4, 5.2.5	5.3.2

Resumen

Se revisaron los detalles de implementación de las clases que componen la arquitectura propuesta, se desarrollaron tres casos en los que se usa el Framework para controlar un robot real, un robot virtual y una aplicación de escritorio. También se presento la evidencia de los resultados obtenidos.

En el siguiente capítulo se presentan las conclusiones, logros alcanzados y el trabajo futuro.

Capítulo 6 Conclusiones y trabajo futuro

En esta sección del documento, se presentan los logros alcanzados en este trabajo de tesis en base a los objetivos planteados. De igual forma se proponen algunos trabajos a futuro que pudieran desarrollarse a partir de los conocimientos reunidos en esta investigación.

6.1 Conclusiones

6.1.1 Respecto al objetivo general

El objetivo general planteado en la sección 1.1.1, es Diseñar una arquitectura para el control y teleoperación de robots CNC, compuesto por un Framework para el desarrollo de aplicaciones de control de robots CNC y un protocolo de comunicación para la teleoperación.

Para dar cabal cumplimiento, se realizó un cuidadoso análisis del estado del arte en el Capítulo 2, donde se pudo conocer el enfoque actual para resolver el problema de la operación a distancia de los robots CNC y se evaluaron las tecnologías candidatas para proponer una solución. Se aprovecharon los conceptos de ingeniería de software para analizar y diseñar la arquitectura que se presento en el Capítulo 4. La arquitectura se llevó a la implementación en el Capítulo 5, donde además se aplicó a diferentes casos de estudio, y se realizaron pruebas de su funcionamiento.

A partir de lo cual se puede concluir que el objetivo general se alcanzó satisfactoriamente.

6.1.2 Respecto a los objetivos específicos

Objetivo 1. Diseñar un Framework para el desarrollo de aplicaciones de control de robots de control numérico.

El Framework es un componente de software creado para desarrollar aplicaciones, que encapsula las funciones de control y teleoperación para robots CNC. Su análisis y diseño se realiza en el Capítulo 4. El modelo específica puntos de extensión que deberán ser desarrollados cuando se aplique a un tipo de robot en particular.

Objetivo 2. Diseñar un componente para el control de robots CNC, que interprete programas RS274-D.

A partir del análisis y diseño realizado en el Capítulo 4, se procede a realizar su implementación en código en el párrafo 5.2.2.2, durante esta etapa se desarrollan dos nuevos algoritmos para el procesamiento de líneas y arcos en un espacio discreto, y se menciona el aspecto de la resolución que puede ser obtenida por el robot. En este componente se implementan los algoritmos para el procesamiento de los comandos RS274-D.

Objetivo 3. Diseñar un protocolo de comunicación para la teleoperación de robots CNC.

Tomando como directriz los casos de uso descritos en el Capítulo 4, en la sección 5.2.2.3 se implementan las instrucciones requeridas para construir el protocolo de comunicación y la forma de operación del esquema cliente-servidor para la teleoperación de robots CNC.

Objetivo 4. Desarrollar un controlador para el robot MAXNC-10 a partir del Framework.

Este trabajo surge en el Grupo de Robótica y Análisis de Imágenes como consecuencia de un problema real, el software proporcionado por el fabricante también tiene las limitaciones mencionadas en la sección 1.3 en el planteamiento del problema. El desarrollo de un controlador para este robot, permitirá que pueda ser usado en sistemas operativos más modernos y desde varios equipo conectados a distancia.

Objetivo 5. Desarrollar una aplicación de escritorio que permita la edición, simulación y ejecución de comandos RS274-D, construida usando el Framework.

El resultado global del trabajo de tesis, se puede resumir en la aplicación de escritorio que permite el acceso a todos los componentes de la Framework por medio de una interfaz gráfica. Para el desarrollo de esta aplicación, el programador solamente usó los métodos proporcionados por el Framework, logrando un código limpio y fácil de leer. Su descripción se muestra en la sección 5.2.5 como aplicación a un caso particular.

A partir de lo anterior, se concluye que los objetivos específicos se alcanzaron de acuerdo a lo esperado.

6.1.3 Trabajo futuro

La investigación desarrollada realiza la integración de diferentes herramientas metodológicas y tecnológicas, sin embargo la necesidad de contar con soluciones para la teleoperación de robots controlados numéricamente ha crecido drásticamente gracias a la popularidad de Internet y la necesidad de integrar los procesos de manufactura con los procesos administrativos, por lo cual el trabajo desarrollado puede ser usado como base para los siguientes trabajos a futuro.

A partir del Framework

- Extender el marco de trabajo para su integración en los sistemas de información empresariales, para compartir y reutilizar segmentos de programas CNC y obtener retroalimentación con la información generada por el robot.

Respecto al módulo de control

- Se pueden desarrollar analizadores sintácticos que interpreten otros lenguajes de programación CNC.
- Agregar funciones para obtener información del robot que pueda ser usado para la teleoperación y su integración con otros sistemas.

A partir del módulo de teleoperación

- Se puede agregar un componente que permita visualizar remotamente el trabajo que esta realizando el robot en tiempo real.
- Agregar funciones de diagnóstico remoto, por ejemplo reportar fallas hacia dispositivos móviles.
- Construir una versión basada en objetos distribuidos, para ser usado en equipos de cómputo con mayor poder de procesamiento.

6.1.4 Comentarios finales

Se construyó un marco de trabajo para el desarrollo de sistemas de tele-operación y control de robots CNC, comprensible por personas con diferentes niveles de dominio de las herramientas de programación.

Uno de los motores más importantes en el desarrollo económico de nuestro país es el avance tecnológico, en este aspecto los robots CNC desempeñan un papel importante. Este trabajo trata de aportar con un granito de arena, esperando que pueda ser de utilidad para aquellos empresarios, investigadores y estudiantes que tengan la necesidad de operar un robot CNC a distancia.

Este trabajo estaría incompleto si no se menciona que un aspecto muy importante es el desarrollo del hardware, que se puede promover y realizar trabajo conjunto.

Anexos de Teleoperación de robots CNC

Anexo A	Gramática del lenguaje RS274-D
Anexo B	Diagramas de secuencia
Anexo C	Secuencias para el robot MAXNC-10

Anexo A. Gramática del lenguaje RS274-D

Para poder analizar, interpretar y ejecutar un programa escrito en un lenguaje, es indispensable conocer el léxico y sintaxis de dicho lenguaje. A continuación se describe la gramática del lenguaje RS274-D en la forma EBNF [46].

6.1.4.1 Reglas de sintaxis

Una línea de RS274-D debe cumplir los siguientes requisitos para poder concluir que cumple con las reglas de sintaxis:

- Opcionalmente puede iniciar con un identificador de línea,
- A continuación debe tener cualquier número de comandos, parámetros y comentarios.
- Se permiten líneas y caracteres en blanco, pero estos son ignorados
- El lenguaje no es sensible al uso de mayúsculas y minúsculas, en las reglas gramaticales se usan de forma indistinta.
- Cada línea se interpreta una sola vez
- La regla de inicio o entrada al interprete es línea

6.1.4.2 Gramática del lenguaje RS274-D.

```
línea       =
      [número_linea],{segmento},fín_de_línea;
número_linea=
      letra_n,digito,[digito],[digito],[digito],[digito] ;
segmento    =
      {espacio},(comentario | comando | parámetro),{segmento} ;
comando     =
      letra_g|letra_m|letra_s , numero_entero;
parámetro   =
      letras_parametro , numero_real;
(* Se define {…} como cualquier conjunto de caracteres que se    *)
(* encuentren antes del fin de línea o del cierre de comentario. *)
comentario  =
      (línea_comentario,{…},fín_de_línea)
      | (paréntesis_derecho,
        (signo_por,{…},signo_por) | {…},
     paréntesis_izquierdo )
    );
```

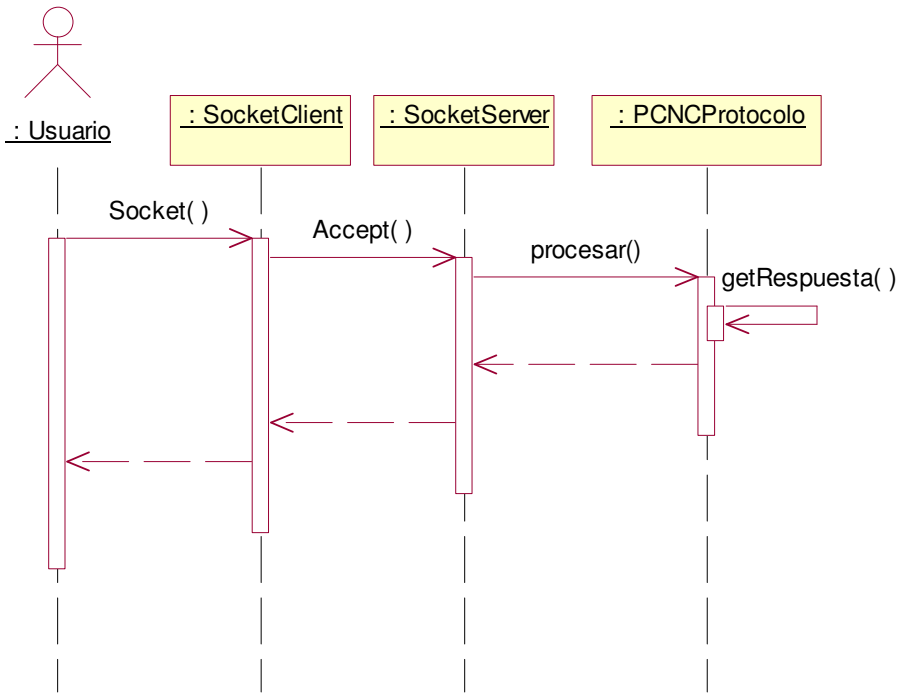
```
número_entero=
      digito,{digito};
número_real =
      [signo],
      ( (digito, { digito }, [punto_decimal],{ digito } )
      | (punto_decimal,digito,{ digito } )
      ) ;
signo       =
      signo_suma | signo_resta ;

(* A continuación se define el conjunto reglas usadas como *)
(* terminadores                                            *)
digito      =
      ‘0’ | ‘1’ |’12 |’3’ |’4’ |’5’ |’6’ |’7’ |’8’ |’9’
letras_parametro=
      ‘a’|’b’|’c’|’d’|‘f’’|’h’|’i’|’j’|’k’|’l’|’p’|’q’|’r’|’t’|’x’|’y’|’z
’
letra_g     =
      ‘g’|’G’
letra_m     =
      ‘m’|’M’
letra_n     =
      ‘n’|’N’
letra_s     =
      ‘s’|’S’
paréntesis_derecho      =
      ‘(‘
paréntesis_izquierdo    =
      ‘)’
linea_comentario  =
      ‘#’
signo_suma        =
      ‘+’
signo_resta       =
      ‘_’
signo_por         =
      ‘*’
signo_div         =
      ‘/’
punto_decimal           =
      ‘.’
espacio                 =
      ‘ ‘|’\t’|’\r’     (*Carácter no imprimible*)
fín_de_línea            =
      ‘\n’        (*Carácter no imprimible*)
```

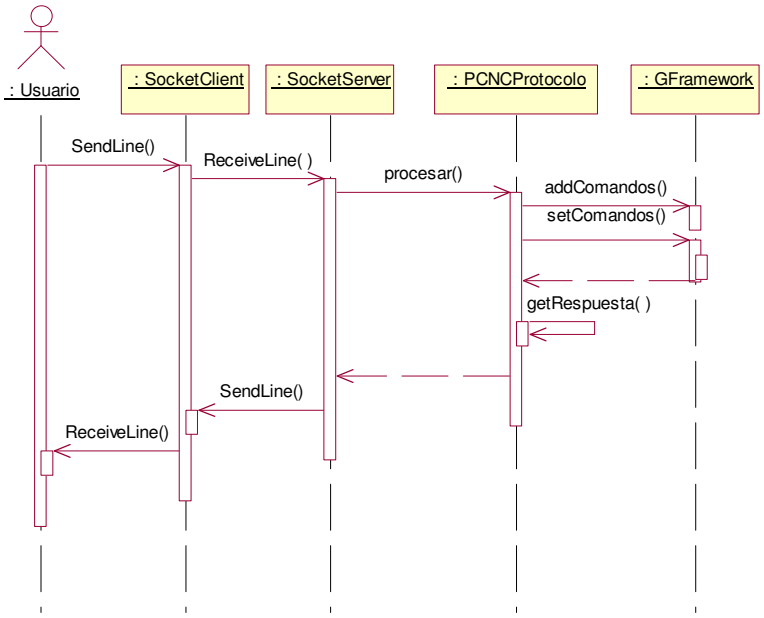
Anexo B. Diagramas de secuencia

6.1.4.3 Diagrama de secuencia del sub módulo: Teleoperación CNC

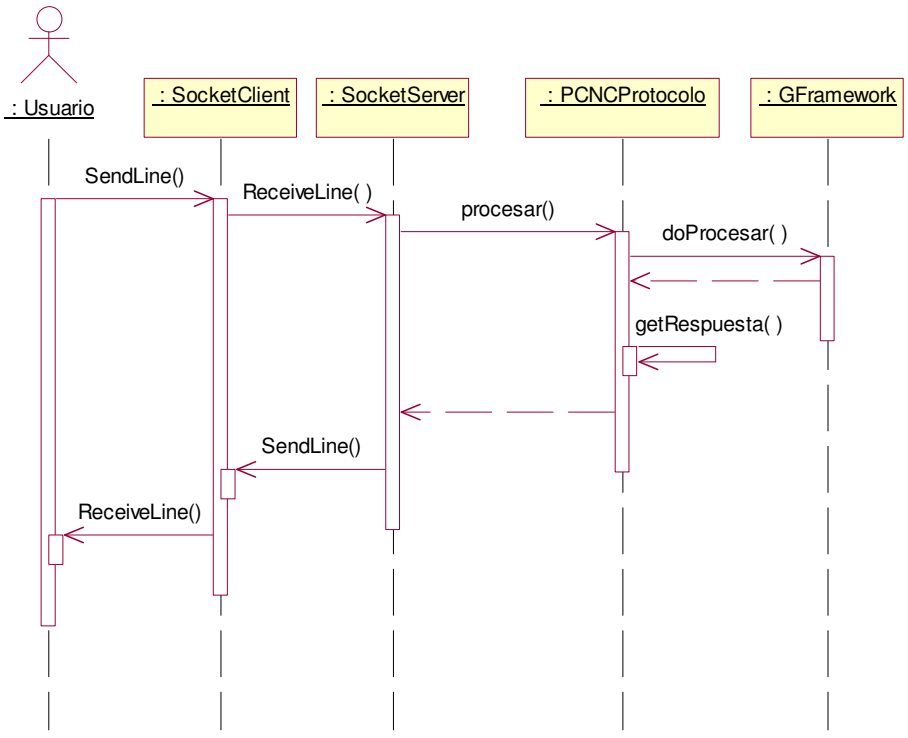
6.1.4.3.1 Diagrama de secuencia: Ingresar



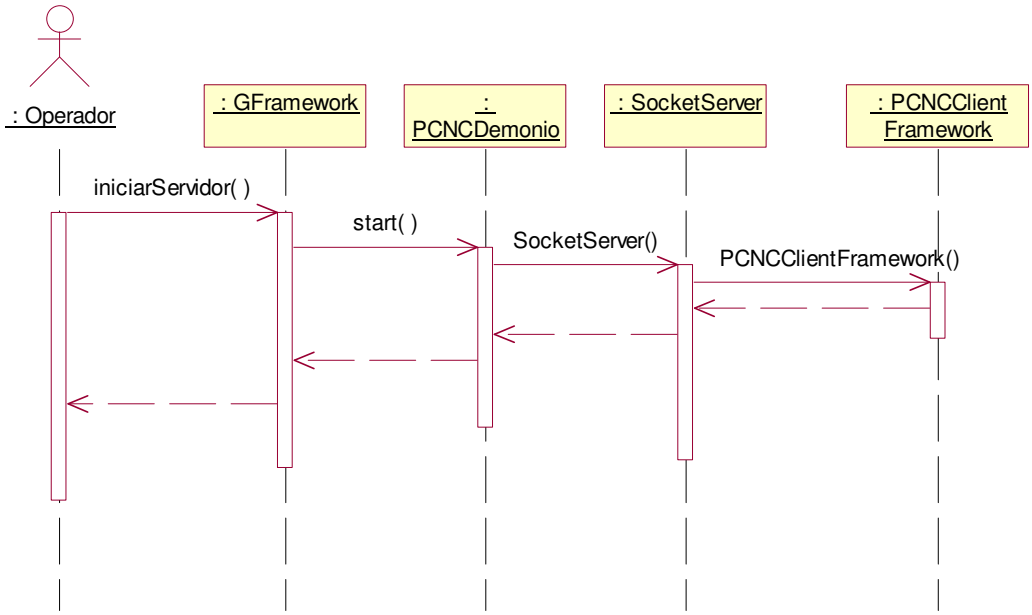
6.1.4.3.2 Diagrama de secuencia: Enviar



6.1.4.3.3 Diagrama de secuencia: Ejecutar

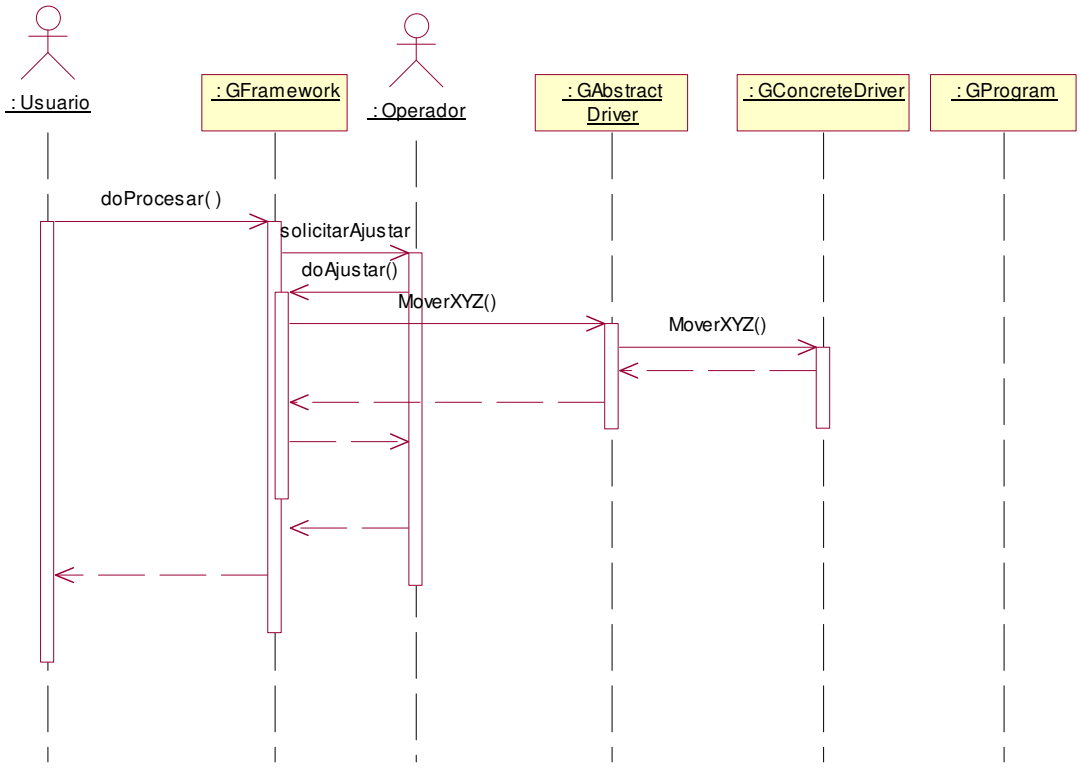


6.1.4.3.4 Diagrama de secuencia: Iniciar/Detener Servidor.

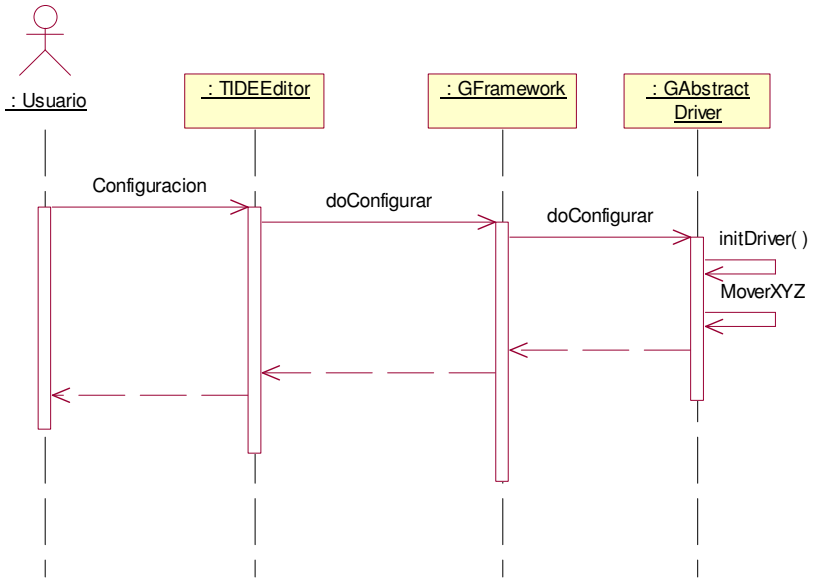


6.1.4.4 Diagramas de secuencia del sub módulo: Control CNC

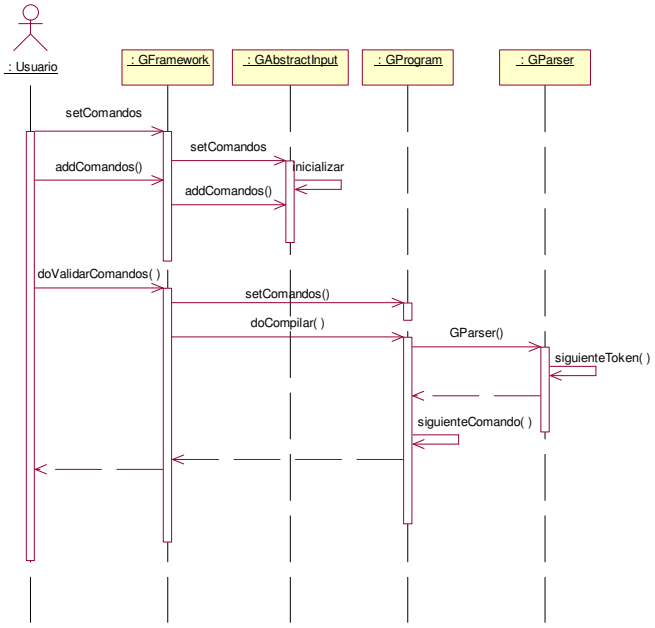
6.1.4.4.1 Diagrama de secuencia: Ajustar



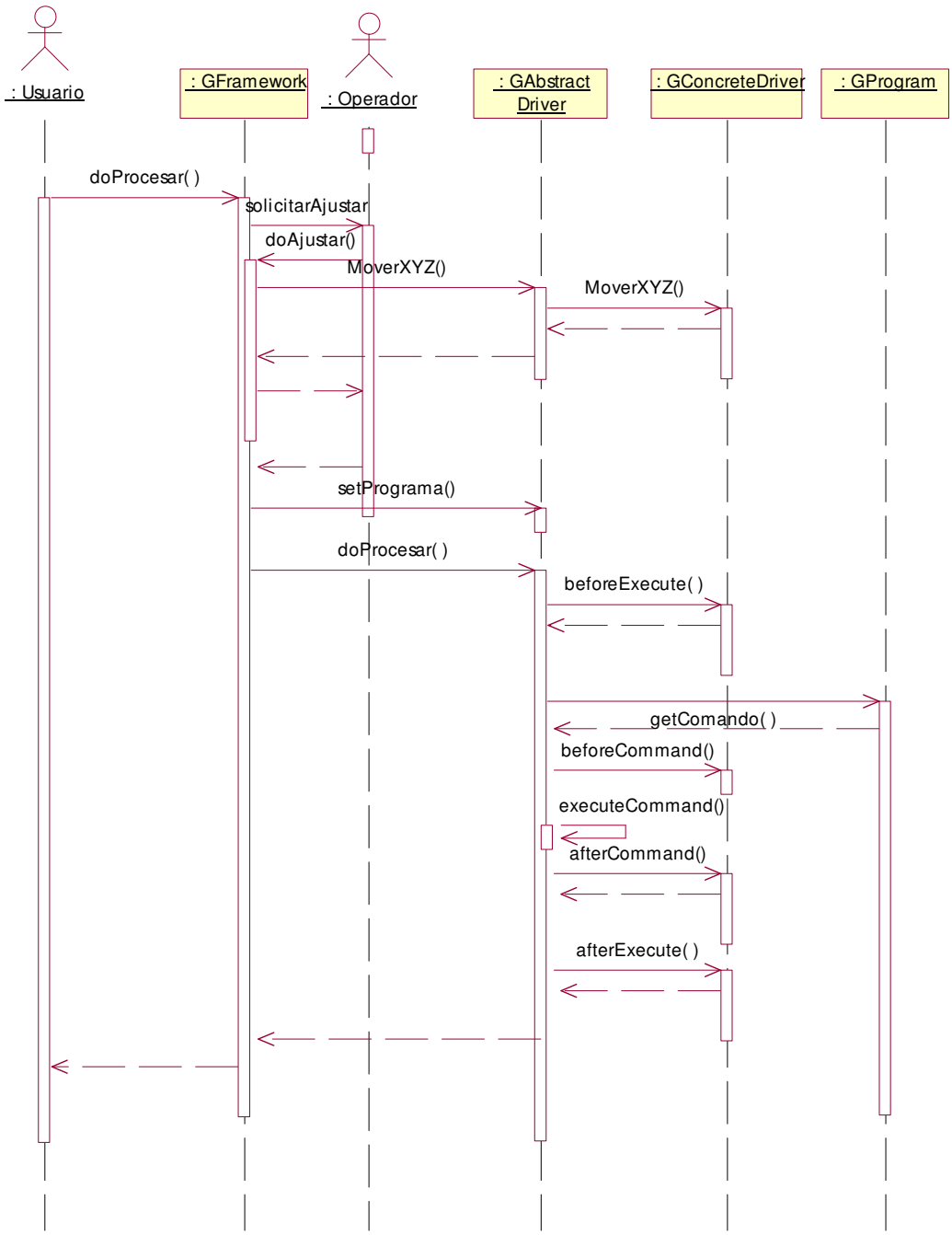
6.1.4.4.2 Diagrama de secuencia: Configurar



6.1.4.4.3 Diagrama de secuencia: Programar



6.1.4.4.4 Diagrama de secuencia: Procesar



Anexo C. Secuencias para el robot MAXNC-10

El fabricante del robot MAXNC-10, define la siguiente secuencia de pasos para obtener el máximo torque de los motores a pasos. Esta no coincide con la mostrada en la Tabla 13, porque la secuencia activa dos y tres motores en lugar de uno y dos, en cada paso.

Secuencias para el Motor X

7	6	5	4	3	2	1	0			On
			XA					D0	XA	1
		XB						D1	XB	1
	XC							D2	XC	1
XD								D3	XD	1
Secuencia de pasos										
1	1	1	0	0	0	0	0	224		
1	1	0	0	0	0	0	0	192		
1	1	0	1	0	0	0	0	208		
1	0	0	1	0	0	0	0	144		
1	0	1	1	0	0	0	0	176		
0	0	1	1	0	0	0	0	48		
0	1	1	1	0	0	0	0	112		
0	1	1	0	0	0	0	0	96		

Secuencias para el Motor Y

7	6	5	4	3	2	1	0			On
							YA	Strobe	YA	0
						YB		AutoF	YB	0
					YC			Init	YC	1
				YD				Selec	YD	0
Secuencia de pasos										
				-0	1	-1	-0	6		
				-0	1	-1	-1	7		
				-0	1	-0	-1	5		
				-1	1	-0	-1	13		
				-1	1	-0	-0	12		
				-1	0	-0	-0	8		
				-0	0	-0	-0	0		
				-0	0	-1	-0	2		

Secuencias para el Motor Z

7	6	5	4	3	2	1	0			On
							ZA	D0	ZA	1
						ZB		D1	ZB	1
					ZC			D2	ZC	1
				ZD				D3	ZD	1
Secuencia de pasos										
				1	1	0	1	13		
				1	1	0	0	12		
				1	1	1	0	14		
				0	1	1	0	6		
				0	1	1	1	7		
				0	0	1	1	3		
				1	0	1	1	11		
				1	0	0	1	9		

Referencias

[1] A.Götz, Et al. (2003). *Tango A Corba Based Control System*. Proceedings of ICALEPCS2003, Gyeongju, Korea.

[2] A.Götz. (1997). *TACO: An object oriented system for PC's running Linux, Windows/NT, OS-9, LynxOS or VxWorks*. PCAPAC Conference , Hamburg,

[3] Andrew S Glassner. (1993). *Graphics Gems*. Ed. Morgan Kaufmann. ISBN 0-122-86166-3. pp. 215-217.

[4] Bertrand Meyer. (1999). *Construcción de Software Orientado a Objetos*. 2ed, Ed. Prentice Hall Madrid. ISBN 84-8322-040-7.

[5] Bruegge, Bernd y Dutoit, Allen H. (2002). *Ingeniería de software orientado a objetos*. Ed. Pearson Educación, México, ISBN 970-26-0010-3. Cp. 9.

[6] Byron Tapley, Ovid W Eshbach. (1990). *Eshbach's Handbook of Engineering Fundamentals*. Ed. Wiley-IEEE, ISBN 0471890847.

[7] C. Goertz. (1963). *Manipulators used for handling radioactive materials.* In E. M. Bennet (Ed.), Human Factors in Technology, Chapter 27, McGraw-Hill, 1963. Chapter 27, Ed. E. M. Bennet, McGraw-Hill.

[8] C.M. Pancerella and R.A. Whiteside. (1996). *A CORBA-Based Manufacturing Environment, Sandia Report SAND96-8557.UC-405.* Sandia Laboratorires.

[9] Charles River Media. (2003). *BSD Sockets Programming from a Multi-Language Perspectiva*. ISBN 1-584-50268-1.

[10] Chris Schroeder. (1996). *Inside Orcad*. Ed. Newnes. ISBN 0750697008, Pg. 186.

[11] Chris Schroeder. (1997). *Printed Circuit Board Design.* Ed. Newnes, ISBN 0750698349.

[12] Cnccontrols. (2005). http://www.cnccontrols.com/. Referencia de Internet.

[13] Dave Shreiner, Mason Woo, Jackie Neider, Tom Davis. (1997). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.1*. Ed. Addison-Wesley, ISBN 0-201-46138-2.

[14] Direct3D architecture (2004). *Microsoft DirectX and XNR, Programming Guide*.

[15] Douglas T. Ross. (1978). *ACM SIGPLAN Notices, The first ACM SIGPLAN conference on History of programming languages*.

[16] Editor. (2005). *Revista. CNC Machining*. Volumen 9, Issue 32, pg 39.

[17] EIA Standard. (1979). *EIA-274-D*, "*Interchangeable Variable, Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines*", Electronic Industries Association, Washington, D.C.

[18] Erick Gamma, Richard Helm, Ralph Johnson, and John Vlissides. (1995). *Design Patterns; Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison-Wesley. ISBN 0-201-63361-2.

[19] Fayad,M.E. Y Schmt, D.C. (1997). *Object Orientes Application Framework*. Communication of the ACM, 40(10):32-38.

[20] Foley, James D., et al. (1995). *Computer Graphics: Principles and Practice. Second ed*. in C. Reading, MA: Addison-Wesley.

[21] Frank Buschmann, Régine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stahl. (1996). *Pattern-Oriented Software Architecture—A System of Patterns.* New York, NY: John Wiley and Sons, Inc.

[22] Fuhua LIN, Cheuk Lun HON & Chuan-Jun -S U. (1996). *A Virtual Reality-based Training System for CNC Milling Machine Operations*. Hong Kong, Annual Journal of IIE.

[23] G.W. Vickers with R.G. Oetter. (1998). *Numerically Controlled Machine Tools*. Ed. Prentice Hall. ISBN 0-13-625526-4.

[24] Geoff Williams. (2003). *Cnc Robotics*. Ed. McGraw-Hill Professional. ISBN 0071418288.

[25] Henry Sowizral, Kevin Rushforth and Michael Deering, (1997). *The Java(TM) 3D API Specification*. Ed. Addison Wesley, ISBN: 0201325764.

[26] ISO/IEC 14977:1996 Extended BNF.

[27] Ivar Jacobson, Grady Booch, James Rumbaugh. (2000). *El Proceso Unificado de Desarrollo de Software*. Ed. Pearson Educación. ISBN 84-7829-036-2.

[28] James Madison. (1996). *CNC Machining Handbook*. Ed Industrial Press Inc. ISBN 0831130644.

[29] Janet Louise Axelson. (1996). *Parallel Port Complete*, Ed. lakeview research llc, ISBN 0-965-08191-5.

[30] Kay A. Robbins Steven Robbins. (1997). *UNIX Programación Práctica.* Ed. Prentice Hall ISBN 968-880-959-4. Pgs 491,516.

[31] Kennedy, John. *Bresenham integer Only Line Drawing Algorithm*. Santa Monica College, Santa Monica, CA 90405, rkennedy@ix.netcom.com.

[32] Klafter, R. D., Chmielewski, T. A., and Negin. (1989). *M. Robotic Engineering: An Integrated Approach*. Prentice Hall, Englewood Cliffs, NJ.

[33] Kris Jamsa, Ken Cope. (1996). *Programación en Internet*. Ed. McGrawHill, ISBN: 970-10-0989-4.

[34] M. Eugene Merchant, Senior Consultant. (2003). *An interpretive review of 20th century us machining and grinding research*. TechSolve, Inc.Cincinnati, Ohio, USA.

[35] Mark Segal Kurt Akeley. (2004). *The OpenGL® Graphics System: A Specification,* Silicon Graphics, Inc.

[36] Merchant, M.E. (1961). *The Manufacturing-System Concept in Production Engineering.* Research, Annals of the CIRP, Vol. 10, pp. 77-83.

[37] Michael Owings, Dennis Clark. (2002). *Building Robot Drive Trains*. McGraw-Hill Professional, ISBN 0071408509.
[38] Mike Mattson. (2001). *CNC Programming Principles and Applications*. Ed. Thomson Delmar Learning, ISBN 0766818888.
[39] Mr. Cox of California, chairman. United States Congress, House Report 105-851.
[40] Omac architecture working group. (2002). *OMAC baseline architecture*. Omac working group.
[41] Omac architecture working group. (1999). *Bussines justification of open architecture control, white paper version 1.0*. Omac users group.
[42] OpenCNC Software, Manufacturing Data Systems.
[43] OSE Consortium. (1998). OSEC-II Project Technical Report. Pp 7,23,17.
[44] P. Lutz and W. Sperling, (1997). *OSACA - the vendor neutral Control Architecture*. Proceedings of the European Conference on Integration in Manufacturing IiM'97, ISBN 3-86005-192-X.
[45] Parsons, J.T. and Stulen, F.L. (1958). *Motor Controlled Apparatus for Positioning Machine Tool*. U.S. Patent No. 2,821,187.
[46] R. S. Scowen. (1996). *Extended BNF A generic base standard*. National Physical Laboratory, Teddington, Middlesex, UK.
[47] Ray Henry, Dan Falk et al. (2003). The Enhanced Machine Control User Handbook, LinuxCNC.org. pp. 7-8.
[48] Reintjes, J.F. (1991). *Numerical Control - Making A New Technology*. Oxford University Press, New York.
[49] S. A. Brown, c. E. Drayton, b. Mittman. (1963). *A Description of the APT Language*.
[50] Sinan Si Alhir. (1998). *UML in a Nutshell.* Ed O'Reilly. ISBN 1565924487. pp 129-172, 215-216.
[51] TAYLOR, K. and Trevelyan, J. P. (1995). *Australia's telerobot on the web*. 26th Symposium on Industrial Robotics, Singapore, pp 39-44. http://telerobot.mech.uwa.edu.au/l.
[52] W.Bolton. (2001). Mecatrónica Sistemas de Control Electrónico en Ingeniería Mecánica y Eléctrica. Ed. Alfaomega. ISBN 9-701-50635-9. pp 178-184.
[53] William Grosso. (2001). Java RMI. Ed. O'Reilly, ISBN 1-565-92452-5.
[54] Wyard-Scott, R.Frey, Q.H.M.Meng. (1998). A robotic Internet workStarion design paradigm. Proceedings of the Third ASCE Specialty Conference on Robotics for Challenging Environments. New Mexico. pp 251-257.

Glosario

Framebuffer. Es una componente de hardware o dispositivo virtual que forma parte de un entorno gráfico. Es un área de memoria que sirve para almacenar información requerida para dibujar y procesar una imagen antes de ser enviada a la pantalla. Una figura puede ser representada por una matriz de colores en dos dimensiones. Esta matriz es implementada en memoria de acceso rápido. Está compuesto por diferentes buffers que complementan la información de la figura. [3].

Framework. Es un diseño reutilizable de todo o parte de un sistema, representado por un conjunto de clases abstractas y la forma en la cual sus instancias interactúan [19].

Interface de Programación de Aplicaciones (API. Application Program Interface) Es un grupo de funciones que usan los programadores a fin de desarrollar programas de aplicación para un ambiente de cómputo específico.

IP. El Protocolo de Internet, es un protocolo de la capa de red que mueve la información entre computadoras anfitriones.

Middleware. Capa intermedia de comunicación que permite que las aplicaciones funcionen en un ambiente de red.

MS-DOS. Sistema operativo de línea de comandos, sin capacidad de multitareas, obsoleto para la mayoría de aplicaciones.

Paquete. Conjunto de clases relacionadas entre si agrupadas en una estructura común.

Resolución. Es el cambio mínimo del valor de entrada capaz de producir un cambio observable en la salida [52].

Protocolo. Es un conjunto de reglas y convenciones aceptadas para comunicar dos entidades [33].

Renderizado. Es el proceso por el cual una computadora genera una imagen sintética, con sus luces, efectos y objetos, a partir de una descripción matemática.

TCP. El Protocolo de Control de Transporte, es un protocolo de la capa de transporte que mueve la información entre aplicaciones orientado a conexión.

Token. Unidad mínima de un lenguaje, tiene tipo y valor.

UDP. El Protocolo de Control de Transporte, es un protocolo de la capa de transporte que mueve la información entre aplicaciones orientado a no conexión.

UML (Unified Modeling Lenguaje). El Lenguaje Unificado de modelado, es un estándar para especificar, visualizar, construir y documentar los artefactos de un sistema de software, negocios y sistemas ajenos al área de software.

Acrónimos

API Application Programming Interface.
JNI Java Native Interface.
NIST Nacional Institute of Standards and Technology.
ORB Object Request Broker, conocido también como middleware.
RMI Remote Method Invocation.
UML Unified Modeling Lenguaje.
IDL Interfaz Definition Language.
OMG Object Management Group.