UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE INGENIERIA ELECTRICA Y ELECTRONICA

SISTEMA DISTRIBUIDO PARA SIMULACION Y CONTROL DE PROCESOS

T E S I S

PARA OBTENER EL TITULO PROFESIONAL DE :

INGENIERO ELECTRONICO

RUBEN ANTONIO DIAZ CANCHAY

PROMOCION: 1996-I

LIMA - PERU

1,997

DEDICATORIA

Mi agradecimiento total y absoluto al impulsor principal de este proyecto, el asesor y amigo PhD Luis Herrera Bendezú sin cuya paciencia y colaboración no hubiese sido posible este trabajo.

SUMARIO

Diseñar e implementar un Sistema de Software Distribuido portable, usando el modelo Cliente/Servidor, para Simulación y Control de Procesos; ejecutandose sobre la plataforma del Sistema Operativo UNIX usando el lenguaje de programación C++, con una interface al usuario totalmente gráfica sobre el sistema de ventanas X Window. Esta tesis presenta el desarrollo de un Sistema de Software Distribuido basado en tecnicas de análisis y diseño orientado a objetos y sistemas Cliente/Servidor.

SISTEMA DISTRIBUIDO PARA SIMULACION Y CONTROL DE PROCESOS

ÍNDICE

PROLOGO

La tendencia mundial en la industria de manufactura y procesos es hacia una minimización de los costos de producción, optimización de recursos, minimización de los tiempos de producción, y aumento de la calidad de los productos. Una manera de enfrentar parte de estos retos que impone la industria moderna es usando las técnicas de la tecnología de la información y las nuevas teorías de control.

Los Sistemas de Control Distribuido (SCD) empezarón como reemplazo del panel de instrumentos. Pero rápidamente fuerón evolucionando hasta convertirse en una red de información de planta, computación y control.

A finales del 80 y principios del 90 los SCD evolucionan a sistemas en red. Los estándares en software y comunicaciones hicieron posible la inter-operabilidad entre diferentes plataformas de hardware y aplicaciones de software lo cual fue absorvido en los SCD. Estos estándares incluyen:

- Sistemas Operativos abiertos UNIX/POSIX
- Modelo de comunicación basado en Sistemas Abiertos Interconectados, OSI
- Protocolo X-Windows para comunicación entre estaciones de trabajo
- Base de Datos Distribuidas
- Accseso via SQL a Base de Datos
- Programación Orientada a Objetos y lenguajes independientes de la plataforma

Los SCD son organizados generalmente en 5 subsistemas. Estos son: (1) Estación de Operaciones, (2) Red de Comunicaciones, (3) Controlador, (4) Procesamiento de Datos, y (5) Adquisición de Datos. La Estación de Operación es la consola donde el usuario visualiza la operación del sistema. La información es eminentemente gráfica y se desarrolla usando X Windows[10] y XView[25].

El SCD opera en una red de computadoras que se conectan usando una red de comunicaciones. Para nuestro sistema utilizamos computadoras con sistema operativo UNIX en una red ethernet con protocolo TCP/IP.

En esta primera fase los procesos a controlar no son procesos físicos, sino modelos de simulación. Para ello se ha desarrollado el modelo matemático de dos plantas: péndulo invertido y tanque doble. En la segunda fase del trabajo se podrá reemplazar el modelo de simulación por una planta real.

El controlador es el elemento que genera las señales que actuan sobre la planta de tal forma que esta se comporte de acuerdo a las especificaciones dadas. El diseño del controlador involucra métodos de la teoría de control clásica y moderna, control óptimo, e identificación de sistemas y control adaptivo. Este segmento del proyecto esta orientado a la investigación en el area de control.

El Procesamiento de Datos involucra la manipulación de la información para gerenciar el sistema, análisis estadistico, y supervisión. Estas funciones no son parte de este trabajo por lo cual no serán implementadas.

Finalmente, la Adquisición de Datos corresponde a la forma de adquirir la información. Para esta fase del proyecto, este subsistema se reduce al sistema de comunicaciones via mensajes ya que las plantas son simuladas.

CAPÍTULO I
CONCEPTOS PRELIMINARES

1.1 Sistemas de Control Distribuido

Los Sistemas modernos de Control Distribuido de Procesos distribuyen las operaciones de monitoreo, supervisión, y control en varias computadoras conectadas en una red de comunicaciones.

1.1.1 Evolución de los Sistemas de Control Distribuido

La tranformación iniciada en los años 50's en el área de control de procesos, con la introducción de un computador digital, experimentó una paulatina evolución en función de los progresos tecnológicos de aquellos dispositivos. En los ultimos años, sin embargo, la irrupción de los microcircuitos digitales VLSI han precipitado cambios bruscos y sustanciales en la concepción global de los sistemas industriales de instrumentación y control.

Los SCD's basados en microprocesadores aparecierón a mitad de los 70's. Inicialmente fueron concebidos como reemplazos para los paneles de instrumentación electrónica. Los primeros sistemas utilizaron displays numéricos. Estos sistemas evolucionaron rápidamente adicionando estaciones de trabajo con video y controladores compartidos con múltiples opciones de control donde predominaba el control PID.

La disposición de estas unidades de cómputo conduce a un nuevo enfoque en el diseño físico de los equipos. El poder de computo no reside en un "sistema" conexo a los dispositivos de instrumentación y control sino que puede ser considerado un "componente" constitutivo de los mismos, un elemento activo más, dotado sin embargo de un grado de flexibilidad que ofrece nuevas posibilidades y perspectivas al proyecto. Esta circunstancia determina una característica arquitectural en los sistemas, que invierte la tendencia original de concentración de la capacidad de cómputo en una gran máquina central responsable de múltiples de tareas. Es de ese modo que una acción compleja sobre un proceso se descompone en un número de acciones simples, asignando un dispositivo de cómputo a cada una de ellas.

Esta estructura caracterizada por la utilización de múltiples procesadores, denominada "inteligencia distribuida" impone la consideración de dos aspectos vinculados entre si:

- Interrelación de los elementos procesadores.

- Normalización de los subsistemas.

La interrelación mencionada, en lo que se refiere a procesadores, significa transferencia de información. Dicha transferencia requiere un medio y un protocolo adecuados. Si consideramos además que una planta industrial compleja presenta una estructura constituida por gran número de elementos, distribuidos sobre superficies que pueden ser extensas, dos tipos de vinculación serán necesarias:

- Corta distancia e interracción moderada.
- Larga distancia e interacción débil.

En un esquema de interacción moderada, cada procesador opera sobre recursos propios pero dispone de una interface hacia un medio compartido, a tráves del cual se podrán definir recursos globales utilizables por los diferentes procesadores. El medio compartido es generalmente de transferencia paralela y por ende apto para cortas distancias. El acoplamiento débil por su parte, implica alguna forma de conexión serie como medio a tráves del cual diferentes procesadores pueden intercambiar mensajes y/o datos.

La realización práctica de ambos tipos de interconexión está vinculada con el problema de la normalización de los componentes del sistema. El empleo de estructuras estandar para las comunicaciones permite una importante reducción de costos de instalación en sistemas de instrumentación y control, al mismo tiempo que aumenta la flexibilidad del conjunto al permitir el intercambio de elementos dentro del sistema.

Al madurar la industria de las PC's a mediados de los 80's se facilitaron las cosas para quien deseara desarrollar un paquete económico de scaneo, control, alarma y adquisición de datos (SCADA) sobre la plataforma de una PC e integrar con este, paquetes de software de propósito general, programas como una hoja de cálculo, de desarrollo publicitario o manejo de base de datos y uno podría tener alternativas efectivas de bajo costo para los SCD [1].

A causa de las limitaciones de las PC's disponibles, estos SCD tenian un bajo nivel de seguridad en aplicaciones consideradas criticas. Fue entonces que los proveedores de SCD's sintieron la necesidad de enriquecer su caja de herramientas para manejar aplicaciones de control de procesos en tiempo real, pero incorporando en sus sistemas:

- Manejadores de Base Datos Relacionales [2].
- Paquetes de hojas de cálculo.
- Capacidad de Control de Procesos Estadisticos.
- Sistemas Expertos [3].

- Simulación de Procesos basados en computadora.
- Diseño asistido por computadora.
- Desarrollo publicitario.
- Manejo de pantalla orientado a objetos.
- Estaciones de trabajo orientadas a ventana.
- Intercambio de información con otros sistemas.

A finales de los 80's y principios de los 90's, los SCD evolucionan a sistemas de red de información de planta, información y control. Los estandares en software y comunicaciones hicieron posible la interoperabilidad entre diferentes plataformas de hardware y aplicaciones de software lo cual fue absorbido en los SCD. Estos estandares incluyen:

- Estandares de Sistemas Operativos Abiertos, tal como UNIX o POSIX.
- Modelo de Comunicación basado en Sistemas Abiertos Interconectados (OSI) [4].
- Modelo de computo cooperativo Cliente/Servidor [5].
- Protocolo X-Window para comunicación gráfica entre estaciones de trabajo.
- Sistemas de Manejo de Base de Datos distribuidas.
- Acceso de SQL [6] a Base de Datos Relacionales Distribuidas.
- Programación Orientada a Objetos y lenguajes independientes de la plataforma.
- Ingenieria de Software asistida por computadora.

1.1.2 Arquitectura de un Sistema de Control Distribuido

Los SCD's son organizados generalmente en 5 subsistemas. Estos son :

1. Estación de Operaciones.
2. Subsistema de Control.
3. Subsistema de Adquisición de Datos.
4. Subsistema de Computo de Procesos.
5. Subsistema de Comunicaciones.

Una arquitectura típica de tercera generación es mostrada en la figura 1.1.

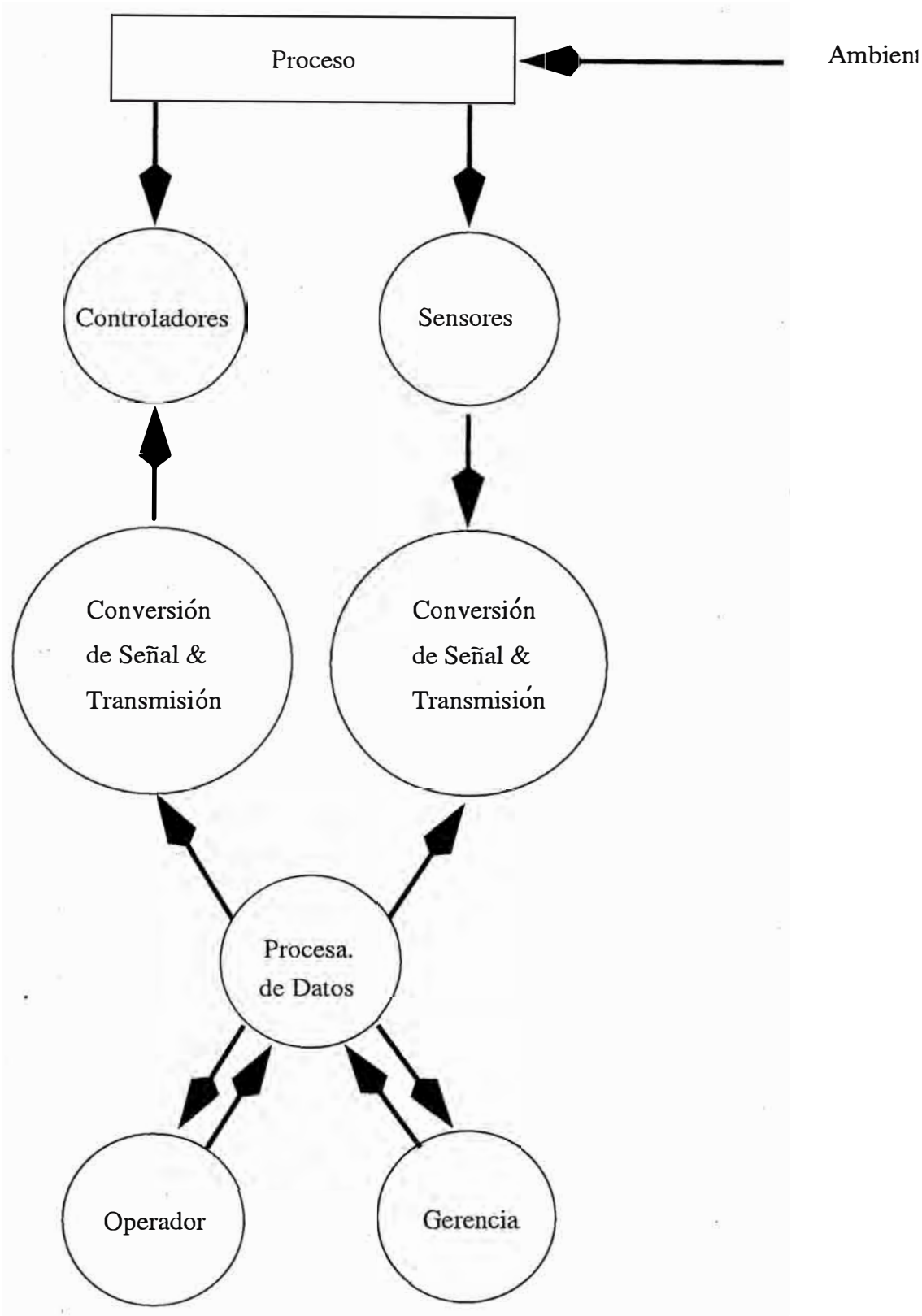

Figura 1.1: Arquitectura de un Sistema de Control Distribuido

Estación de Operaciones

El criterio primario para las primeras estaciones de trabajo fue la aceptación del operador; para lo que se propuso representaciones en video de los procesos como representaciones de los paneles de instrumentación. Estos videos fueron organizados en diferentes vistas, para permitir al operador interactuar con el video de manera similar a las operaciones del panel. Las ventajas de esto fueron tanto operacionales como económicas (mayor flexibilidad en el arreglo de displays, reducir el tamaño de la sala de control, pocos operadores y bajar costos de diseño e instalación).

Las pantallas fueron organizadas en jerarquías donde la pantalla entera en un nivel podía representarse sólo por una pequeña porción de la pantalla en el siguiente nivel de la jerarquía.

Fueron añadiendose procesos gráficos después al repertorio de pantallas, ofreciendo representaciones de procesos ó la sección de un proceso con información del proceso actualizada en él. Los teclados fueron usados para manipular estos displays. Su diseño fue poco innovativo con utilización de teclas de función para las pantallas.

Mientras un panel de instrumentación permite al operador ver todos los instrumentos a la vez; en una estación de trabajo con video es casi imposible mostrar en una sola pantalla toda la información requerida. Para compensar esto, aparecieron lámparas como alarmas auxiliares. Ellas representan areas específicas de la planta y podrían alertar al operador por las condiciones mostradas en la pantalla. Estos anunciadores físicos pueden luego ser reemplazados para cubrir areas dedicadas en la pantalla en si, lo cual se manifiesta en cambio de colores para indicar una alarma en una sección de una planta que no esta siendo mostrada en ese momento. Una vez alertado el operador puede buscar la alarma y tomar alguna acción. Varias tecnicas innovativas fueron ofrecidas en este sentido, una fue ofrecer un botón para cada anunciador, una segunda fue tener el sistema buscando a tráves de la jerarquia de pantallas y presentar la pantalla asociada con la condición de alarma cuando un botón de alarma fuera presionado.

Algunos sistemas habilitan al usuario a priorizar las alarmas tal que las más importantes sean vistas primero. Las variables individuales pueden cambiar su prioridad basados en la naturaleza de la alarma o en el tiempo que la variable permanece en condición crítica.

Un punto donde los SCD destacan es en su capacidad para proporcionar información clara de los procesos que se estan ejecutando. Los medios de almacenamiento magnético que disponen estos sistemas podían almacenar información de miles de variables a la vez. Cuando las estaciones de operaciones evolucionan llegan a proporcionar toda la información necesaria para que los operadores regulen sus procesos y puedan tomar

decisiones tacticas, como cuando empezar un proceso o como ajustar los parametros de otro.

Las estaciones de trabajo orientadas a ventanas permiten que la pantalla de la estación de operaciones sea dividida en secciones escalables, con cada subsección actuando como si fuera un subsistema independiente. Conforme al protocolo X Window la información puede aparecer en ventanas independientes separadas en la misma pantalla. Los usos más avanzados pueden combinar información de diferentes sistemas para ofrecer un mosaico en una sola pantalla cubriendo todos los aspectos del proceso. Además, la información en pantalla puede ser enviada a cualquier estación conectada a la red que realize la solicitud. Esto permite que varias estaciones de trabajo puedan ver la misma ventana al mismo tiempo. Asi todas las estaciones tienen acceso total e independiente a toda la información y aplicaciones en el ámbito de la red.

Subsistema Controlador

Este subsistema es el encargado de generar las señales que actuan sobre la planta de tal forma que esta se comporte de acuerdo a las especificaciones dadas. El diseño del Controlador involucra métodos de la teoría de control clásica y moderna, control óptimo, e identificación de sistemas y control adaptivo.

Subsistema de Adquisición de Datos

Estos subsistemas involucran la manipulación de la información para gerenciar el sistema, realizar un análisis estadistico y supervisión. Reemplazan a los paneles de indicadores y registros, aceptando una variedad de señales. La información es transmitida a tráves de la red del SCD y reportada a varios subestaciones incluyendo estaciones de trabajo. Una vez en el sistema, la información puede ser despachada a muchas locaciones como se necesite y estaciones de trabajo geograficamente separadas pueden mostrar la misma información simultaneamente. La información puede ser utilizada por el subsistema de control de procesos, para el cálculo de consumo de energía, tasas de producción, eficiencia.

Durante los 80's los subsistemas de colección de datos fueron gradualmente desplazados por subsistemas distribuidos de entrada/salida. Estos subsistemas son más inteligentes y funcionales que sus predecesores. Algunos módulos incluyen capacidades programables tales como la conversión de la señal del proceso a unidades de ingeniería. Alguna funcionalidad del subsistema de Control como rutinas de cálculo las cuales pueden ser ejecutadas muy frecuentemente o requieren de una respuesta muy rápida para las condiciones de un proceso han migrado del subsistema de Control al subsistema

de entrada/salida.

Subsistema de Computo de Procesos

La diferencia principal entre los SCD y los paneles convencionales es el sistema de interface, ya que un SCD usa toda la información que esta disponible en la red vía un cable de conexión de alta velocidad. Dada la gran cantidad de información que se envía a la computadora del proceso se busca siempre minimizar los costos de instalación. Dentro de este contexto aparecieron los gateways que permiten a otras computadoras ser conectadas al SCD. El gateway es el traductor de información entre la computadora y el sistema SCD.

Como el subsistema de Control llega a ser más poderoso y funcional, el rol del subsistema de computo involucra ejecutar supervisión de procesos de alto nivel, procesamiento de información y análisis. El rol de supervisión involucra un proceso a coordinar donde el análisis y procesamiento de información necesario es substancial, como es el caso del desarrollo de Sistemas Expertos.

Entre las tareas de procesamiento se incluyen las siguientes

- Estrategias basadas en el modelo de optimización.
- Aplicaciones de Sistemas Expertos.
- Análisis estadistico de la calidad de producto y proceso.
- Manejo de registros electrónicos, selección de variables del proceso y eventos, alarmas del proceso y condiciones extremas, acciones del operador y supervisor, observaciones y notas del operador.

1.1.3 Herramientas de Sistemas de Control Distribuido

1. **Manejo de Base de Datos Relacionales**

   Los paquetes de manejo de Base de Datos Relacionales se han convertido en el pilar de los sistemas de información modernos. Ellos organizan la información dentro de tablas, cada una conteniendo listas de información, llamadas registros. Cada registro contiene un conjunto de datos referidos como campos. Esta estructura relativamente simple es accesada y manipulada por lenguajes especializados tales como Structured Query Language (SQL).

   Estos lenguajes permiten a la información dentro de las tablas ser combinada entre ellas, ordenadas y seleccionadas para soportar virtualmente cualquier aplicación. La simplicidad de la estructura facilita ser modificable para los requerimientos de

la aplicación. Esta posibilidad es probablemente la razón por la que se ha convertido en el almacenamiento de información estandar.

Los paquetes RDBMS[1] y más recientemente versiones distribuidas o de red han encontrado aplicación en dos areas mayores dentro del entorno de los SCD. La primera es el almacenamiento de información en la construcción de paquetes configurables tal como estrategias de control y rutinas de colección de datos. El valor principal en estos tipos de aplicaciones es que todas las referencias a información especifica estan en un lugar común y alli un cambio hecho en una figura es reflejado en todos las otras figuras que usen la misma información. El segundo es el registro histórico de cualquier información relacionada a los procesos. La ventaja de los RDBMS en esta clase de aplicaciones es la posibilidad de seleccionar los registros apropiados de diferentes tablas basado en un criterio de selección y presentar la información resultante para uso en un reporte, una pantalla u otra aplicación, como una hoja de cálculo.

2. **Diseño Asistido por Computadora**

   Los paquetes CADD[2] han sido usados hace más de una década para diseño de aplicaciones mecánicas y eléctricas. Entre sus características está el aplicarlas junto a otros programas de ingeniería para aplicar varias condiciones al modelo para determinar gráficamente como el objeto creado responderia a condiciones del mundo real. Los CADD combinados con software de manejo de Base de Datos proporcionan una efectiva plataforma para el diseño y configuración de estrategias de control en SCD's.

3. **Desarrollo Publicitario**

   A pesar que la mayoria de SCD's no utiliza software de automatización de oficina, el uso de herramientas de Desarrollo Publicitario es importante, incluyendo procesadores de palabra, herramientas de presentación gráfica, herramientas de formato de documentos y todo lo que sea de uso práctico dentro de la oficina.

4. **Ingeniería de Software Asistida por Computadora**

   Las herramientas de Ingeniería de Software Asistida por Computadora proporcionan un entorno para los usuarios para desarrollar, diseñar, evaluar y modificar sus aplicaciones. Este ambiente incluye:

   - Un entorno de desarrollo integrado, completado con un editor de fuentes y un evaluador de programas.

[1] Sistema Manejador de Base de Datos Relacionales
[2] Computer Aided Design

- Una libreria de rutinas reusables que han sido certificadas y evaluadas para su uso.
- Una facilidad "make" la cual construye programas ejecutables desde módulos disponibles en la libreria.
- Herramientas de manejo de documentos para seguir la revisión de programas asi como de documentos de soporte, tales como especificaciones funcionales, diseño de documentos e instalación y manuales de usuario para asegurar la compatibilidad del desarrollo del software.

Las herramientas CASE[3] orientadas a objetos prometen grandes ganancias en productividad. Ellas actualmente estan ganando aceptación en el mercado para SCD's. La programación orientada a objetos ha sido aplicada con enfásis en el desarrollo de paquetes de dibujo gráfico donde objetos "complejos"son creados desde objetos más simples y una vez creados son manipulados como una entidad simple, movidos a través de la pantalla, rotados o cambiando de color. Los objetos fueron incorporados primero dentro de SCD's como herramientas de construcción de gráficos estáticos, pero ellos rápidamente evolucionaron para soportar objetos dinámicos cuyos atributos fueron relacionados a variables del proceso. Cambios al valor del proceso puede cambiar los atributos del objeto mostrado como el color, longitud, posición.

La encapsulación de información en tiempo real y procedimientos dentro de objetos se ha extendido para incluir tratamientos de datos, representación de dispositivos, rutinas de análisis. El uso de tecnicas orientadas a objetos ha simplificado el esfuerzo requerido para diseñar sofisticadas herramientas para estaciones de trabajo.

5. **Hojas de Cálculo**

Los paquetes de hoja de cálculo son usados por personal de ingeniería y personas de negocios a quienes les permite resolver relaciones aritméticas o lógicas con datos ordenados como una tabla dividida en celdas. Los resultados del cálculo de la fórmula son mostrados en la celda correspondiente. Una vez entrada la información puede ser movida, copiada u operada por varios comandos construidos en la hoja de cálculo. Las macros, colecciones de comandos ejecutados por invocación de una tecla, puede ser entrada en celdas de la hoja de cálculo para simplificar su uso en aplicaciones complejas.

Los SCD utilizan las hojas de cálculo para importar información histórica y luego

[3]Computer Aided Softwae Engineering

operarla por formulas almacenadas o macros para ejecutar una variedad de aplicaciones en los procesos de análisis. Muchos paquetes de hoja de cálculo están siendo diseñados para trabajar en conjunto con los RDBMS para aumentar las capacidades de ambos.

6. **Control de Procesos Estadisticos**

   Los paquetes de Control de Procesos Estadisticos son tipicamente usados para guiar al personal de operaciones y de ingeniería en el continuo aumento de la calidad de un producto a tráves de sucesivos refinamientos en el proceso de producción.

7. **Software de Sistemas Expertos**

   Los Sistemas Expertos permiten al usuario impartir un conocimiento base dejando el proceso especificado por cientos a miles de reglas y relaciones. Las reglas interactuan con cada otra y con información de los procesos. Además como son capaces de asimilar y considerar muchos factores, son referidos como módulos de Inteligencia Artificial. Las aplicaciones donde Sistemas Expertos han sido aplicados en un entorno de control de procesos son:

   - Diagnóstico de causas de eventos anormales tales como alarmas multiples.
   - Selección de la mejor de entre muchas posibilidades correctas.
   - Interpretación de estructuras de instrucciones perdidas tales como comandos de voz aceptados.

8. **Software de Simulación**

   La Simulación ha sido una parte integral del proceso de ingeniería de control y operaciones de procesos por decadas. Los modelos matemáticos de alta precisión usados en aplicaciones tales como Reacciones Quimicas Cineticas y el diseño de procesos requiere los recursos de sistemas de computadora. Algunos simuladores especializados fueron construidos para operaciones de simulación de plantas de energía núclear.
   Hoy las herramientas de Simulación se han convertido en herramientas de propósito general e incorporados dentro de los SCD para usarlos en un rango de aplicaciones que van desde la evaluación de diseño de sistemas de control a módulos para entrenamiento. Cuando son incorporados a los SCD, estos modelos son capaces de ser refinados continuamente por el flujo constante de datos de los procesos disponibles en las unidades actuales.

1.1.4 Redes de Comunicaciones

Las redes de comunicaciones son la contraparte de la instrumentación convencional. Estas permiten integrar los diferentes subsistemas trabajando bajo una base de información común para estar mejor informado y tomar decisiones en menor tiempo.

Las claves en el diseño y evaluación de una red de comunicaciones incluyen pruebas de comunicación, integridad de la información que se mueve de la fuente al destino, comportamiento de la red de comunicaciones en el ambiente (Ej: inmunidad al ruido), métodos de recuperación para retransmitir información, distancia de trasmisión, número de estaciones capaces de enviar o recibir información, tipos de información que se puede transmitir (variables del proceso, mensajes, programas aplicación) y la posibilidad de modificar la topología de la red (adicionar nuevas estaciones por ejemplo) mientras el sistema esta en operación.

Varios estandares de comunicaciones han sido adoptados para su uso en SCD's. Estos incluyen el Protocolo de Control de Transmisión/Protocolo Internet (TCP/IP), que fue inicialmente desarrollado en 1969 en cooperación con el Departamento de Defensa para uso en el proyecto ARPA de los Estados Unidos. Este es el protocolo más usado. Interconecta cientos de entidades de gobierno, universidades, y agencias privadas a tráves del mundo. Además por su amplio uso, reputación para un buen servicio e integración dentro de sistemas y aplicaciones de software (tal como UNIX), es una elección muy común dentro de la comunidad de usuarios.

A mitad de los ochenta la Organización de Estandares Internacionales introdujo una metodología para implementar redes de comunicaciones llamada Interconexión de Sistemas Abiertos (ISO) [4]. Esta fue desarrollada para promover la interoperabilidad entre sistemas dispares y esta siendo adoptada por virtualmente todos los sistemas.

ISO es un modelo de siete capas, que consiste de las siguientes:

1. *Capa Física.* Esta capa se ocupa de la transmisión de bits a lo largo de un canal de comunicaciones. Su diseño debe asegurar que cuando un extremo envía un bit con valor 1, éste reciba exactamente como un bit con ese valor en el otro extremo, y no como un bit de valor 0. Preguntas comunes aquí son cuántos voltios deberán utilizarse para representar un bit de valor 1 o 0; cuántos microsegundos deberá durar un bit; y cual será la asignación de los pines del conector al canal de comunicaciones.

2. *Capa de enlace.* La tarea primordial de la capa de enlace consiste en, a partir de un medio de transmisión común y corriente, transformarlo en una linea sin errores de transmisión para la capa de red.

3. *Capa de Red* La capa de red se ocupa del control de la operación de la subred. Un punto de suma importancia en su diseño, es la determinación sobre cómo encaminar los paquetes del origen al destino. Las rutas podrían ser estáticas y dificilmente cambiadas o dinámicas determinándose en forma diferente para cada paquete. La responsabilidad, para resolver problemas de interconexión de redes heterogéneas recaerá, en la capa de red.

4. *Capa de transporte.* Su función principal consiste en aceptar los datos de la capa de sesión, dividirlos, siempre que sea necesario, en unidades más pequeñas, pasarlos a la capa de red y asegurar que todos lleguen correctamente al otro extremo.

5. *Capa de Sesión.* La capa de sesión permite que los usuarios de diferentes máquinas puedan establecer sesiones entre ellos. A través de una sesión se puede llevar a cabo una transmisión de datos ordinario, tal y como lo hace la capa de transporte, pero mejorando los servicios que ésta proporciona y que se utilizan en algunas aplicaciones.

6. *Capa de Presentación.* La capa de presentación se ocupa de los aspectos de sintaxis y semántica de la información que se transmite. Un ejemplo típico de servicio de la capa de presentación es el relacionado con la codificación de datos (por ejemplo, ASCII y EBCDIC).

7. *Capa de Aplicación.* Contiene una variedad de protocolos que se necesitan frecuentemente. Uno de los servicios que presenta es el de un terminal virtual de red que permite establecer comunicación entre terminales incompatibles. el software completo del terminal virtual se encuentra en la capa de aplicación.

Estas siete capas son un modelo abstracto de como la información podria ser empaquetada para ser enviada desde una aplicación residente en un sistema a una aplicación en una máquina independiente vía una red de comunicaciones. En vez de que estas especificaciones sean unicas para cada capa, se incorpora un conjunto de estandares que son compatibles con el modelo para cada capa. Como resultado hay una gran libertad para implementar el estandar ISO.

1.2 Sistema Operativo UNIX

El sistema operativo UNIX es uno de los mayores avances en el desarrollo de las computadoras. Es un Sistema Operativo poderoso que es independiente de la maquina, con soporte para herramientas de software. El S.O. UNIX proporciona los mismos servicios que todos los otros Sistemas Operativos, permite correr programas, proporciona

una conveniente y sólida interface para una amplia variedad de dispositivos periféricos (impresoras, cintas, discos, terminales, etc) que se conectan a la mayoria de computadoras y proporciona un sistema de ficheros para almacenar información.

Una primera versión del sistema UNIX usando una PDP-11/20 fue patentada por los Laboratorios Bell en 1971. Una de sus características fue la integración de disciplinas hasta ese entonces desarrollandose separadamente como la programación y preparación de documentos.

Pronto el UNIX [7] se convirtió en un Sistema Operativo de propósito general. Posteriormente el UNIX se escribió en lenguaje de programacion C, lenguaje desarrollado especialmente para su trabajo en el Sistema Operativo UNIX. El lenguaje C [8] fue desarrollado para ser adaptable a diferentes tipos de arquitectura de computadoras. Esto sirvio para que el sistema entero pudiera moverse de un entorno a otro con un mínimo de dificultad. Desde entonces el UNIX ha sido transportado a virtualmente cada arquitectura de computadora aparecida en el mercado.

El UNIX es pionero en varios principios importantes; siendo uno de los más grandes la portabilidad. Además uno de sus mayores aportes es el pipe, que da la idea para programar funciones complicadas como un conjunto de programas trabajando juntos. Otra idea asociada al UNIX es la de "Herramienta de Software". Esta idea no es original del Sistema pero ciertamente ha sido la más desarrollada que en otros Sistemas Operativos. La mayoria de las "Herramientas de Software" son pequeños programas que hacen una cosa bien y que pueden cooperar con otras herramientas para ejecutar tareas más sofisticadas.

Ciertas funciones del Sistema Operativo son usadas casi continuamente. Por ejemplo, la parte del sistema UNIX que envuelve la conmutación de un programa a otro (tiempo compartido) es necesitada muchas veces cada segundo. En el S.O. UNIX, todas las funciones que son necesitadas inmediatamente son constantemente almacenadas en memoria. La parte residente en memoria de un Sistema Operativo es llamada el kernel[4]. Otras funciones del Sistema Operativo son usadas sólo ocasionalmente, tal como la posibilidad de mover un archivo a otro lugar. Estas funciones son proporcionadas por utilidades, programas estandar que son invocados por el usuario. En UNIX es fácil adicionar una nueva utilidad escribiendo un nuevo programa.

El kernel del S.O. UNIX es pequeño, de tal manera que la mayoria de funciones del S.O. son proporcionadas por programas utilitarios. Un kernel mínimo puede cumplir con englobar la planificación de tareas, paso de mensajes, memoria virtual, y controladores (drivers) de dispositivos.

El S.O. LINUX [9] es un Sistema Operativo multitarea, multimedia de 32 bit con

[4]Núcleo del sistema

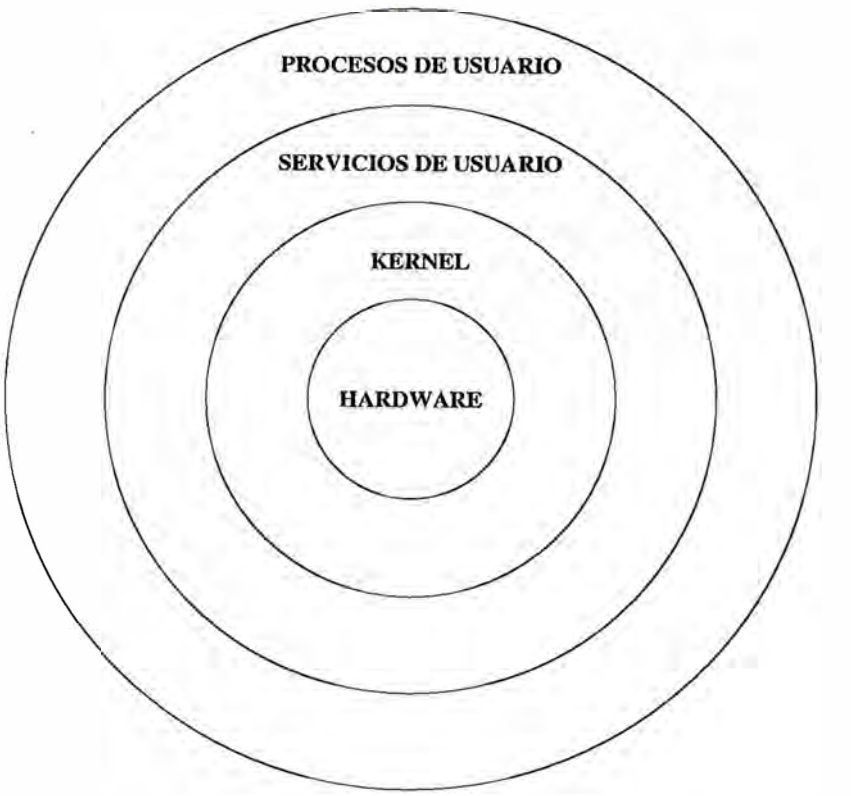

Figura 1.2: Arquitectura de S.O. Linux

código fuente completo desarrollado por la Comunidad de Software Libre en la INTERNET. LINUX es un clone del Sistema Operativo UNIX que corre en computadoras con procesadores Intel 80386/80486/Pentium. Soporta un amplio rango de software desde TeX a X Window System [10], el compilador GNU C/C++, el TCP/IP. El S.O. LINUX es compatible al nivel de fuente con un número de UNIX estandar incluyendo IEEE POSIX, System V y BSD. Linux también proporciona un entorno de programación UNIX completo, incluidas librerias estandar, herramientas de programación, compiladores y depuradores. El S.O. proporciona una interface uniforme a todos los dispositivos periféricos.

La figura 1.2 presenta la arquitectura del LINUX en los términos más generales. La figura 1.3 presenta como los programas de nivel usuario se comunican con el kernel.

El kernel no es una tarea separada bajo LINUX. Es como si cada proceso tuviera una copia del kernel. Cuando un proceso usuario ejecuta una llamada de sistema, no transfiere el control a otro proceso, pero cambia su modo de ejecución desde el modo usuario al modo kernel. En modo kernel, mientras ejecutamos la llamada al sistema el proceso tiene acceso al espacio de direcciones del kernel y por funciones de soporte tiene acceso al espacio de direcciones del usuario ejecutando la llamada.

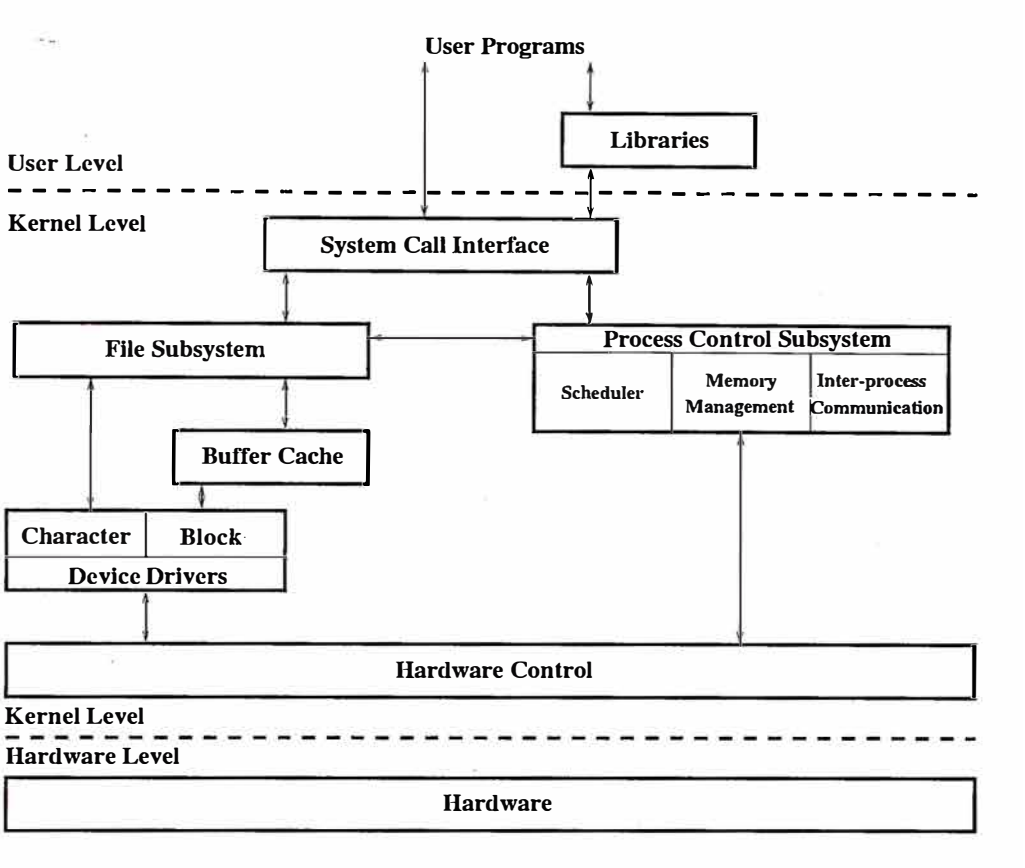

Figura 1.3: Comunicación de programas nivel usuario con kernel de S.O.

1.3 Sistemas de Interface de Usuario Gráficas

Los Sistemas de Interface de Usuario Gráficas (IUG) o Sistemas de Ventana son generalmente construidos usando principalmente dos arquitecturas de tiempo de ejecución: El modelo Cliente/Servidor (ej. X Window) donde se utiliza un Servidor y multiples Clientes; y el modelo CALL-BACK (de llamada-retorno) un modelo de control en el cual las funciones de aplicación son invocadas bajo el control de la IU. También es conocido el modelo MVC (model-view-controller) que es principalmente explotado en aplicaciones SMALLTALK.

Los paradigmas actuales de los sistemas de IUG son:

1. La posibilidad de construir sistemas en los cuales los componentes pueden ser escritos en lenguajes de programación distintos.

2. Componentes que pueden estar ejecutandose simultaneamente en entornos distribuidos, heterogeneos sin compartir espacio de direcciones.

3. Que la arquitectura pueda ser cambiada en tiempo de ejecución.

4. Que multiples usuarios puedan interactuar con el sistema.

5. Varias cajas de herramientas (toolkits) puedan ser empleadas.

Un sistema configurado por componentes y conectores es una arquitectura de IU. Es importante reconocer que esta arquitectura conceptual es distinta de la implementación de la arquitectura. No hay un acuerdo en el número de conectores o componentes que pueden conectarse a un simple conector. Los componentes pueden comunicarse por conectores, la comunicación directa entre componentes no esta permitida y se comunican enviando mensajes; las notificaciones (sucesos o eventos) van hacia niveles inferiores del modelo de la arquitectura de IU y las peticiones hacia niveles superiores de ella. Los conectores son responsables por el enrutamiento y posible multiplexado de los mensajes.

1.4 Principios y Conceptos Fundamentales de los Modelos de Arquitecturas de Interface de Usuario Gráficas

1.4.1 Componentes

Un componente es todo aquel objeto perteneciente a la arquitectura. El objeto puede ser arbitrariamente complejo, como por ejemplo el contenido de una ventana. Sin embargo hay que aclarar que la arquitectura interna de un componente no sólo es un objeto, sino además una interface de diálogo que le permite interactuar con otros componentes dentro del modelo de arquitectura al cual pertenece. Los componentes tienen sus propios módulos de control y tienen un dominio superior e inferior. El dominio superior especifica el conjunto de notificaciones a las cuales el componente responde y el conjunto de peticiones que el componente emite hacia el exterior (hacia niveles mayores del modelo de arquitectura). El dominio inferior especifica el conjunto de notificaciones que los componentes emiten hacia niveles menores de la arquitectura y el conjunto de peticiones a las cuales responde.

1.4.2 Notificaciones y Peticiones

Los componentes en una arquitectura se comunican asincronamente vía mensajes. Los mensajes consisten de un nombre y un conjunto de parámetros asociado. Hay dos tipos de mensajes: notificaciones y peticiones.

Las notificaciones son anuncios de cambios de estado de los componentes. Los tipos de notificaciones que un componente puede emitir son completamente determinadas por la interface de diálogo a los objetos internos de los componentes. Por ejemplo una notificacion puede ser "*una nueva tecla ha sido presionada*", el objeto que genera la notificacion monitorea el uso del objeto interno del componente. El componente que recibe la notificación hace la llamada respectiva a su objeto interno para la actualización respectiva.

Las peticiones por otro lado son directivas de los componentes de las capas inferiores generadas por su interface de diálogo, pidiendo una acción a ejecutar por algún conjunto de componentes de las capas superiores. La respuesta o petición que un componente puede recibir esta determinada por la interface al objeto interno del componente, similar a como las notificaciones son determinadas. La diferencia es que una notificación es el anuncio de que una rutina de la interface fue invocada con los parametros y valores de retorno adecuados, mientras que una respuesta (petición) es el anuncio de la invocación de una de las funciones de acceso del objeto.

1.4.3 Conectores

Los conectores son los que sirven de enlace entre componentes de diferentes niveles de la arquitectura, es decir la presencia de un conector indica la transición de un nivel a otro dentro del modelo de arquitectura. Los conectores pueden ser conectados a cualquier número de componentes como también a otros conectores. La responsabilidad primordial de un conector es el de encaminar los mensajes, una responsabilidad secundaria es el filtraje de mensajes. Las clases de filtraje permitidos son:

- *Sin filtrar:* Cada mensaje es enviado a todos los componentes conectados en el lado relevante del conector (inferior para notificaciones, superior para peticiones).
- *Filtraje de Notificaciones:* Cada notificación es enviada sólo a los componentes que están registrados para ella.
- *Priorización:* El conector define un orden de prioridad sobre los componentes conectados, basándose en un criterio de evaluación especificado por el diseñador de software durante la etapa de construcción de la arquitectura. Los conectores priorizan en casos en los cuales varios componentes conectados a un lado del conector ejecutan la misma función. Como por ejemplo el verificador ortográfico de un documento con posibles diferentes implementaciones. En tales casos es necesario un procedimiento para seleccionar el apropiado destino de la notificación. Por ejemplo un documento HTML necesita un chequeador de palabra que permita ignorar los comandos de cabecera que aparecen dentro de él.
- *Bloqueo de mensaje:* El conector ignora cada mensaje enviado a él. Es usado para aislar subsistemas de una arquitectura y también para progresivamente adicionar componentes a una arquitectura existente. Un desarrollador puede conectar un nuevo componente a una arquitectura y luego poner a funcionar un conector, cambiando el tipo de filtraje cuando el componente esta listo para ser evaluado.

1.4.4 Invocación Implicita

La invocación implicita ocurre cuando un componente reacciona a una notificación y responde invocando algún código. La invocación es implicita debido a que el componente que realizó inicialmente la notificación no sabe si la notificación puede causar alguna reacción.

Muchos sistemas utilizan la invocación implicita para beneficiarse de la separación de módulos, mientras otros aprovechan la invocación implicita para lograr independencia y un orden de componentes.

1.4.5 Mensajes y Mecanismos de Mensajes

Los mecanismos de mensajes existentes transmiten servicios de peticiones, eventos (notificaciones), objetos o una combinación de ellos. Los sistemas actuales se caracterizan por imponer una disciplina en el uso de mensajes y un modelo de uso. El sistema X Window usa un servicio de respuesta y mensajes a eventos, usando sólo un servidor gráfico.

1.4.6 Soporte para Lenguaje y Procesos

Muchos de los sistemas existentes soportan multiples lenguajes, sin embargo la mayoria opta por el uso de un solo lenguaje de programación. Por ejemplo, mientras ahora hay muchos lenguajes diferentes disponibles para el sistema X Window, es díficil el caso que C (y C++) no sean los lenguajes preferidos para el desarrollo en X. Otros sistemas sin embargo no involucran un lenguaje de programación único y tratan de permitir que sus componentes sean escritos en diferentes lenguajes de programación, permitiendo al diseñador la elección del lenguaje de su preferencia; como es el caso del reciente Chiron-2 (C2) [11].

Los sistemas existentes tienden a ser rigidos en su mapeo de procesos. En el otro extremo, las aplicaciones X contienen exactamente dos procesos, un Cliente y un Servidor. Otros sistemas asumen sin embargo espacio de direcciones compartido. Un tercer grupo (GenVOca, Field, SoftBench y C2) [12] satisfacen simultaneamente un arbitrario número de procesos en un espacio de direcciones no compartido que se encuentra disponible.

1.4.7 Modelos de Interoperabilidad entre Componentes

Los modelos de interoperabilidad entre componentes, tales como OLE y OpenDoc [13] proporcionan mecanismos de comunicación estandar para sus componentes. Generalmente, el modelo proporciona un formato para describir los servicios ofrecidos por

un componente y facilidades de tiempo de ejecución para localizar, cargar y ejecutar servicios de otros componentes. Desde que estos sistemas están realizados sobre implementaciones de bajo nivel sirven de poca ayuda para construir sistemas fuera de sus componentes.

1.5 Objetos e Interfaces de Usuario Gráficas

A principios de los 80's aparecieron las primeras GUI's comerciales. Aunque Xerox logro un amplio desarrollo de los sistemas en el mercado en los 70's, la popularidad de las GUI's coincidio con la introducción en el mercado de la Apple Macintosh. Después de un año Microsoft anunció MS Windows, que fue descrito como un entorno de usuario gráfico (aunque no orientado a objetos).

Las GUI's del Sistema Operativo UNIX son orientadas a objetos. MOTIF y OPEN LOOK proporcionan constructores de objetos en pantalla, que son usados por desarrolladores para crear GUI's elegantes (apuntar y presionar). NeXTStep incorpora un sofisticado entorno de objetos para que el usuario maneje fácilmente comandos en linea del UNIX.

1.6 Interfaces de Usuario Gráficas Orientadas a Objetos

Las GUI's orientadas a objetos toman menor tiempo de aprendizaje que los lenguajes en linea y requieren poca o ninguna memorización. Una dificultad de estas interfaces es que son menos flexibles que los lenguajes en linea. Esta deficiencia sin embargo aparece en las implementaciones y no como una limitación teórica. Algunas interfaces sin ser orientadas a objetos permiten al usuario manipular objetos en pantalla con dispositivos como el ratón. Una medida de las GUI's orientadas a objetos es su grado de reutilización. El bloque de construcción básico para una interaface orientada a objeto es el concepto de ventana. En la mayoria de las interfaces gráficas orientadas a objetos hay varios tipos de ventanas. Otros bloques básicos son los menus pop-up y pull-down, el puntero en pantalla, entre otros. Estos tienen un comportamiento similar a otros objetos de pantalla. Ellos también se implementan usando un estilo orientado a objetos por el programador. Los menus pop-up aparecen en pantalla sólo cuando el usuario accede a ellos. El puntero es un objeto controlado por el usuario que responde a los movimientos del mouse o del track-ball. Es usado para controlar y mover otros objetos de la ventana como iconos y menus. El puntero puede tomar varias formas en la pantalla identificando el evento que se esta produciendo.

Los iconos también son una parte importante de las interfaces orientadas a objetos. Son representaciones de ficheros y eventos contenidos dentro del sistema de archivos.

Si un icono representa una aplicación, esta es iniciada cuando el icono es seleccionado. Notar que todos los iconos representan objetos de la vida real. Esta relación hace mas fácil de aprender las aplicaciones.

Combinando estas características de GUI's orientadas a objetos se forma la interface llamada WIMP [14] (windows, iconos, menus y puntero).

1.6.1 La Interface Macintosh

La interface Macintosh es producto de los resultados de investigación en XEROX PARC durante los 70's. El actual Sistema Operativo Apple ha sido escrito en C con ampliación a objetos. La interface Mac es única debido a la relación que permite entre el Sistema Operativo y la computadora en si, pues mucho del código de bajo nivel está ubicado en el hardware de sólo lectura ROM. Estos componentes estan altamente integrados tal que el Sistema Operativo y la interface comparten el mismo nombre. Debido a que la Macintosh no tiene un lenguaje de comandos, todos los accesos al sistema son vía la GUI. La interface Mac mantiene dos objetos primarios: ventanas e iconos.

1.6.2 Microsoft Windows

La interface Microsoft Windows esta basada en tecnología originalmente desarrollada para el administrador de presentacion de OS/2. Las primeras versiones tuvieron pocas características orientadas a objetos. Sin embargo estas caracteristicas fueron superandose hasta conseguir un producto lider en el mercado.

Windows **??** centra su comportamiento alrededor del Programa Manejador de Ventanas donde las aplicaciones son mostradas en ventanas separadas, o grupos; pudiendo el usuario crear grupos y distribuir aplicaciones a ciertos grupos, para facilitar el acceso a ellas.

En la actualidad Windows NT permite correr aplicaciones DOS, Windows, UNIX, y OS/2 usando módulos separados de ejecución (API's) [5] sobre el Sistema Operativo.

1.6.3 X Window, Motif y Open Look

La mayoria de los sistemas de ventana estan basados en kernel; esto es son cerrados al Sistema Operativo en si y pueden correr sólo en un sistema discreto, tal como una estación de trabajo. El X Window no es parte de ningún Sistema Operativo pero en vez de ello está compuesto completamente de programas de usuario. X esta creado bajo el modelo Cliente/Servidor. El sistema esta dividido en dos partes: Servidores de Pantalla

[5] API: Aplication Program Interface

que proporcionan presentación en pantalla y reconocimiento de entrada de usuario y Clientes (programas de aplicación que ejecutan tareas específicas).

El Servidor actua como un intermediario entre programas de aplicación Cliente y el hardware de pantalla local (uno o quizas múltiples pantallas) y dispositivos de entrada (generalmente un puntero o un teclado). Cuando se produce una entrada del teclado o del puntero; el Servidor dirige la entrada a la aplicación Cliente correspondiente. De igual modo, los programas Cliente hacen peticiones (para información, procesos, etc) que son comunicados al hardware de pantalla por el Servidor. Por ejemplo, un Cliente puede requerir que una ventana pueda ser movida o que un texto sea mostrado en pantalla.

La división dentro de la arquitectura X permite a los clientes y al servidor de pantalla trabajar juntos en la misma máquina o residir en máquinas distintas (posiblemente de diferente tipo, con diferentes sistemas operativos, etc) conectadas en una red. Por ejemplo, se podría usar una computadora de bajo poder o una workstation como Servidor de pantalla para interactuar con Clientes ejecutandose en un sistema remoto más poderoso en cuyo caso toda la entrada de usuario y display puede ocurrir en el Servidor de la estación de trabajo existiendo comunicación a tráves de la red usando el protocolo X.

Alguien podría elegir correr un Cliente en una máquina remota por un buen número de razones. Casi siempre la máquina remota ofrece una posibilidad no disponible en la máquina local; una arquitectura mejor desarrollada, un procesador más poderoso y eficiente, software de aplicación diferente. X permite tomar ventaja de estas posibilidades remotas y ver los resultados en el terminal local.

La diferencia entre Clientes y el Servidor permite enfrentar complicadas situaciones de pantalla. Por ejemplo se puede acceder a varias máquinas simultaneamente. X logra mostrar salidas en varios pantallas simultaneamente. Esta característica puede ser muy provechosa en labores educativas. Hipoteticamente un profesor, podría enseñar material educativo a un grupo de estudiantes, cada uno en una estación de trabajo o terminal conectado a la red. Otra ventaja menos obvia del modelo Cliente/Servidor es que ya que el Servidor tiene completa responsabilidad de interactuar con el hardware, sólo el programa Servidor podría estar en una máquina especifica. Programas Cliente X pueden entonces ser fácilmente tranportados de un sistema a otro.

X Window es el manejador de ventanas estandar del Sistema Operativo UNIX, no es orientado a objetos sin embargo; proporciona la mayoria de productos gráficos para Unix. Open Desktop, de AT&T y Sun Microsystems, fue el primer X Window en el mercado. Esta actualmente disponible en workstations de Sun y AT&T. Las principales características de Open Desktop [10] son incluir el "pushpin", que permite al usuario

cerrar aplicaciones abiertas y la caja de dialogo con un sofisticado "zoom" de 3D.

Motif [15] fue producto del desarrollo de la interface de usuario de tecnologia HP, Microsoft y Digital Equipment Corporation (DEC) y derivada de la GUI NewWave. Motif permite a los usuarios reconocer claramente cuando un botón es presionado o una ventana es seleccionada. Motif también forma la base para el Open Desktop de SCO, para la implementación en PC del UNIX System V de AT&T. Open Desktop permite al Motif incluir un administrador de archivos, un paquete de Dos, el administrador de Base de Datos Ingres y el X. Open Desktop es la más popular de las versiones gráficas de UNIX en el mundo de PC's y es clave para mover los sistemas multiusuario basados en texto al mundo Cliente/Servidor.

1.6.4 El Servidor de Pantalla X

El Servidor X es un programa que maneja toda la entrada proveniente de dispositivos, tales como el teclado y mouse, y la dirije a la aplicación Cliente correspondiente, el Servidor también maneja toda la salida de cualquier Cliente que este ejecutándose y actualiza la pantalla al reflejar la salida. Cada pantalla física (que puede tener múltiples pantallas) tiene sólo un programa Servidor.

La entrada de usuario tiene otros tipos de información que pasan del Servidor al Cliente en la forma de "eventos". Un evento es un paquete de información que le informa al Cliente de algo que actualizar, tal como la entrada de teclado. Moviendo el puntero o presionando una tecla se causa un evento de entrada.

Cuando un programa Cliente recibe un determinado evento, responde con una petición al Servidor para realizar una serie de acciones que afecten la pantalla. Por ejemplo, el Cliente puede requerir que una ventana sea dimensionada a un tamaño particular. El Servidor responde por un programa Cliente para actualizar la ventana apropiada en la pantalla correspondiente. Los servidores están disponibles para muchos tipos de sistemas, incluyendo PC's, workstations, y terminales X que pueden tener el Servidor cargado desde otra maquina o almacenado en una ROM.

1.6.5 Clientes

Como mencionamos previamente, un Cliente es un programa de aplicación. La versión estandar de X del MIT incluye más de 50 programas Cliente que ejecutan una variedad de tareas. X te permite correr varios Clientes simultaneamente; cada Cliente mostrado en una ventana diferente. Por ejemplo, se podría editar un texto en una ventana, compilando el archivo fuente de un programa en una segunda ventana, leer el correo en una tercera.

Asi como los Clientes X generalmente muestran sus resultados y toman su entrada desde un Servidor, ellos pueden estar corriendo en una computadora diferente de una red. Otra cosa a notar es que los programas Cliente pueden verse de manera diferente en diferentes servidores ya que X no tiene una interface estandar, además que los usuarios pueden manejar el entorno para aplicar sus Clientes X de manera diferente en cada servidor, sin contar que el hardware del monitor de cada Servidor puede ser diferente.

Asi como un Cliente necesita comunicarse con el Servidor, también necesita hacerlo con otros Clientes. Por ejemplo un Cliente puede necesitar decirle al administrador de ventanas donde ubicar un icono. La comunicación entre Clientes es facilitada por propiedades. Una propiedad es una pieza de información asociada con una ventana y almacenado en el Servidor. Las propiedades son usadas por los Clientes para almacenar información que otros Clientes podrían necesitar saber, tal como el nombre de la aplicación asociado con una ventana particular. Almacenar las propiedades en el Servidor hace que la información que contengan se encuentre accesible a todos los Clientes. El uso más común de las propiedades en la comunicación entre Clientes, es cuando el Cliente le dice al Administrador de Ventanas el nombre de la aplicación asociada con una ventana. Por defecto el nombre de la aplicación corresponde al nombre del Cliente, pero más de un Cliente podría permitir especificar un nombre alternativo cuando se corre el programa.

Un Cliente especial es el administrador de ventanas [6] el cual maneja la posición y tamaño de las ventanas de las aplicaciones ejecutándose en un servidor. El administrador de ventanas es sólo otro cliente, pero por convención, tiene una responsabilidad especial para mediar en la competencia de aplicaciones por demandas de recursos físicos incluyendo espacio de pantalla, color y el teclado. El administrador de ventanas permite al usuario mover ventanas alrededor de la pantalla, redimensionarlas y comenzar nuevas aplicaciones.

Las aplicaciones requieren asimismo dar al administrador de ventanas cierta información que lo ayude a decidir la competencia por demanda de recursos o espacio en pantalla. Por ejemplo, una aplicación puede especificar el tamaño por defecto y su incremento de tamaño.

[6] *window manager*

CAPÍTULO II
MODELOS MATEMÁTICOS DE LOS PROCESOS

2.1 Péndulo Invertido

2.1.1 Fundamento Teórico

El modelo de sistemas mecánicos se puede obtener utilizando la Segunda Ley de Newton para movimiento lineal y rotacional. Esta establece que para un sistema de N partículas $m_1, m_2, \ldots, m_i, \ldots, m_N$ se cumple la siguiente relación:

$$\sum_{i=1}^{N} m_i \ddot{\vec{r}}_i = \sum_{j=1}^{M} \vec{F}_j \tag{2.1}$$

donde:

m_i → masa de la i-ésima partícula

$\vec{r}_i$ → posición del centro de masa de la i-ésima partícula

$\vec{F}_j$ → j-ésima fuerza aplicada al sistema de partículas

Similarmente, para el movimiento rotacional en el plano alrededor de un punto P en un *sistema de referencia inercial* se cumple la siguiente relación:

$$\sum_{i=1}^{N} I_i \ddot{\theta}_i = \sum_{j=1}^{M} \tau_j \tag{2.2}$$

donde:

I_i → momento de inercia de la i-ésima partícula respecto al punto P

θ_i → ángulo recorrido por la i-ésima partícula alrededor del punto P

τ_j → j-ésimo torque aplicado al sistema de partículas

2.1.2 Modelo Matemático del Péndulo Invertido

El sistema Péndulo Invertido consiste en una varilla con pivote montada sobre un carrito móvil con desplazamiento horizontal, tal como se muestra en las figura 2.1. Este es un problema clásico de control donde el objetivo es mover el carrito aplicando una fuerza F de tal forma que el péndulo permanezca vertical. Este es el mismo principio utilizado en naves aeroespaciales que utilizan propulsión ubicada en la base de la nave. Los modelos que se desarrollan corresponden a los sistemas siguientes:

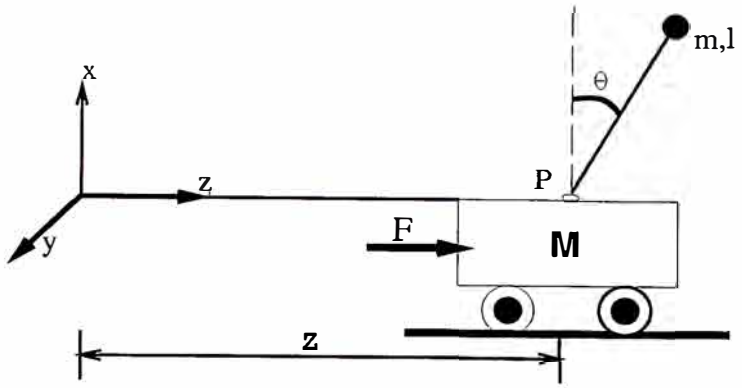

Figura 2.1: Péndulo Invertido de Bola Controlado por Fuerza

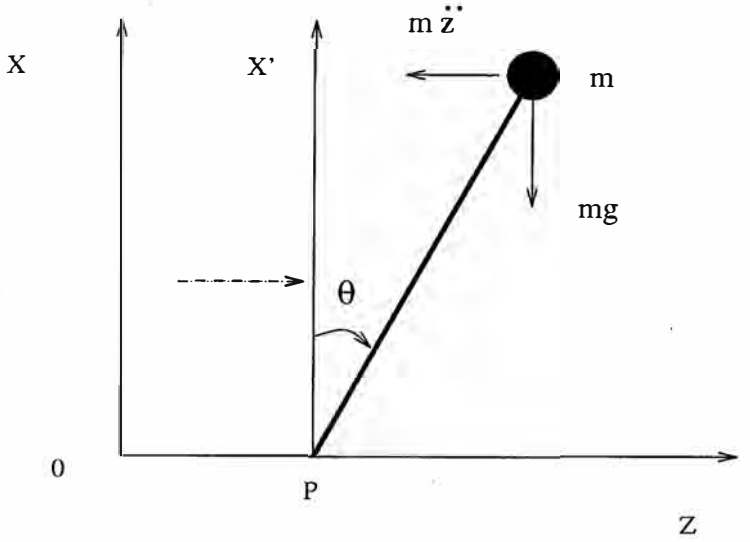

Figura 2.2: Diagrama de Cuerpo Libre para el Sistema PIBF

- Péndulo Invertido de Bola Controlado por Fuerza
- Péndulo Invertido de Varilla Controlado por Fuerza

Por simplicidad estos sistemas se denominan : PIBF y PIVF respectivamente.

Modelo de Sistema PIBF

Para este caso tenemos que $m_1 = M$ y $m_2 = m$, las masas del carrito y péndulo respectivamente. De la figura 2.2, se tiene que la posición con respecto al sistema $z - x$ del carrito de masa M es $(z, 0)$ y la del péndulo de masa m es $(z + lsin(\theta), lcos(\theta))$. De la ecuación 2.1 aplicada al movimiento lineal en la dirección z se tiene:

$$M\frac{d^2}{dt^2}z + m\frac{d^2}{dt^2}(z + lsin\theta) = F$$

desarrollando las derivadas tenemos:

$$(M+m)\ddot{z} + (ml)[-(sin\theta)\dot{\theta}^2 + (cos\theta)\ddot{\theta}] = F. \tag{2.3}$$

Para completar el modelo se utiliza la ecuación 2.2 aplicada al movimiento rotatorio alrededor del punto P, ver figura 2.2. La ecuación 2.2 es válida para un sistema de referencia inercial; lo cual no es cierto en el presente caso debido a la aceleración del sistema $z' - x$, que contiene al punto P, respecto al sistema de referencia principal $z - x$. Para evitar éste problema utilizamos la tercera ley de Newton que permite reemplazar el efecto de la aceleración del sistema de referencia $z' - x'$ por una fuerza $m_i\ddot{z}$, aplicada al centro de gravedad de las partículas i de dicho sistema en dirección opuesta a la aceleración. Ahora podemos decir que el sistema $z' - x'$ es inercial y proceder a usar la ecuación 2.2 para el movimiento rotacional. De la figura 2.2 tenemos:

$$I\ddot{\theta} = mglsin\theta - m\ddot{z}lcos\theta$$

donde I: momento de inercia de la masa m respecto al punto P, $I = ml^2$. Reordenando términos se tiene:

$$m\ddot{z}lcos\theta + (ml^2)\ddot{\theta} = mglsin\theta \tag{2.4}$$

De las ecuaciones (2.3) y (2.4) obtenemos la ecuación de estado de este sistema. La asignación de variables de estado es la siguiente: $x_1 = z$, $x_2 = \dot{z}$, $x_3 = \theta$, $x_4 = \dot{\theta}$:

$$\dot{x_1} = x_2 \tag{2.5}$$

$$\dot{x_2} = \frac{F - mgsinx_3cosx_3 + mlsinx_3x_4^2}{M + msin^2x_3} \tag{2.6}$$

$$\dot{x_3} = x_4 \tag{2.7}$$

$$\dot{x_4} = \frac{(M+m)gsinx_3 - mlsinx_3cosx_3x_4^2 - Fcosx_3}{l(M + msin^2x_3)} \tag{2.8}$$

Este conjunto de ecuaciones puede agruparse en una ecuación de estado de la forma:

$$\dot{x} = f(x, u) \tag{2.9}$$

donde:

$x = [x_1, x_2, x_3, x_4]^T$ es el vector de variables de estado
u es la señal de control (F en este caso) y
f es una función vectorial igual a $[f_1, f_2, f_3, f_4]^T$, donde f_i es igual a $\dot{x_i}$ para $i = 1...4$.

La linealización de la ecuación 2.9 es necesaria para su análisis con métodos de control lineal. Esta linealización [16] se efectua alrededor del punto $x = [0, 0, 0, 0]^T$.

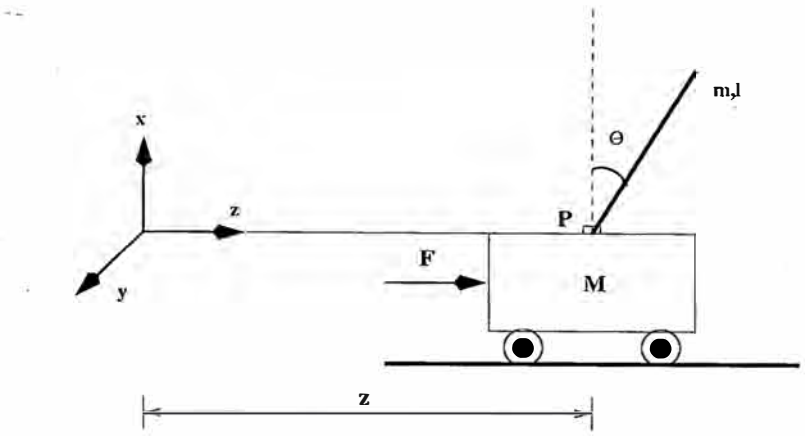

Figura 2.3: Péndulo Invertido de Varilla Controlado por Fuerza

Luego, la ecuación (2.9) linealizada es:

$$\dot{x} = f(0,0) + \frac{\partial f(0,0)}{\partial x}x + \frac{\partial f(0,0)}{\partial u}u$$

$$\dot{x} = f(0,0) + \begin{bmatrix} \frac{\partial f_1(0,0)}{\partial x_1} & \frac{\partial f_1(0,0)}{\partial x_2} & \frac{\partial f_1(0,0)}{\partial x_3} & \frac{\partial f_1(0,0)}{\partial x_4} \\ \frac{\partial f_2(0,0)}{\partial x_1} & \frac{\partial f_2(0,0)}{\partial x_2} & \frac{\partial f_2(0,0)}{\partial x_3} & \frac{\partial f_2(0,0)}{\partial x_4} \\ \frac{\partial f_3(0,0)}{\partial x_1} & \frac{\partial f_3(0,0)}{\partial x_2} & \frac{\partial f_3(0,0)}{\partial x_3} & \frac{\partial f_3(0,0)}{\partial x_4} \\ \frac{\partial f_4(0,0)}{\partial x_1} & \frac{\partial f_4(0,0)}{\partial x_2} & \frac{\partial f_4(0,0)}{\partial x_3} & \frac{\partial f_4(0,0)}{\partial x_4} \end{bmatrix} (x-0) + \begin{bmatrix} \frac{\partial f_1(0,0)}{\partial u} \\ \frac{\partial f_2(0,0)}{\partial u} \\ \frac{\partial f_3(0,0)}{\partial u} \\ \frac{\partial f_4(0,0)}{\partial u} \end{bmatrix} (u-0) \quad (2.10)$$

que se reduce a:

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{(M+m)g}{lM} & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{1}{lM} \end{bmatrix} u \quad (2.11)$$

Esta ecuación tiene la siguiente forma:

$$\dot{x} = Ax + Bu \quad (2.12)$$

donde A y B son las matrices constantes correspondientes de la ecuación (2.11).

Modelo de Sistema PIVF

De la ecuación (2.1) aplicada al movimiento lineal del sistema de la figura 2.4 en la dirección z se tiene:

$$M\frac{d^2}{dt^2}z + m\frac{d^2}{dt^2}(z + (l/2)sin\theta) = F$$

desarrollando las derivadas tenemos:

$$(M+m)\ddot{z} + (ml/2)[-(sin\theta)\dot{\theta}^2 + (cos\theta)\ddot{\theta}] = F \quad (2.13)$$

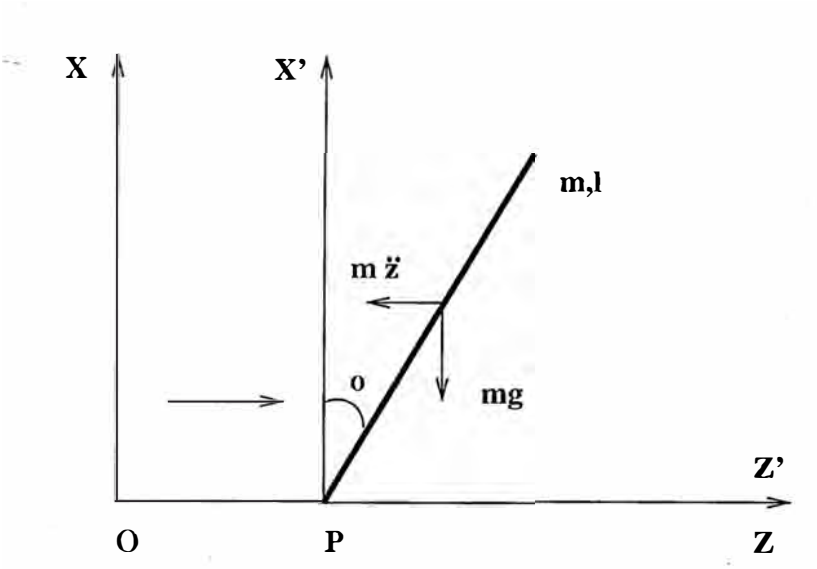

Figura 2.4: Diagrama de Cuerpo Libre para el Sistema PIVF

Al igual que en el caso anterior se aplica la segunda ley de Newton al movimiento rotatorio alrededor del punto fijo P en el sistema de la figura 2.4 con eje de coordenadas $Z' - X'$ que se mueve junto con el carrito. Así obtenemos:

$$I\ddot{\theta} = mg(l/2)sin\theta - m\ddot{z}(l/2)cos\theta$$

donde I: momento de inercia de la varilla de masa m respecto al punto P, $I = ml^2/3$. Reordenando términos se tiene:

$$m\ddot{z}lcos\theta + (ml^2/3)\ddot{\theta} = mglsin\theta \tag{2.14}$$

De las ecuaciones (2.13) y (2.14) obtenemos la ecuación de estado de este sistema. La asignación de variables de estado es la siguiente: $x_1 = z$, $x_2 = \dot{z}$, $x_3 = \theta$, $x_4 = \theta$:

$$\begin{aligned}
\dot{x_1} &= x_2 \\
\dot{x_2} &= \frac{4F - 3mgsinx_3cosx_3 + 2mlsinx_3x_4^2}{4(M+m) - 3mcos^2x_3} \\
\dot{x_3} &= x_4 \\
\dot{x_4} &= \frac{6(M+m)gsinx_3 - 3mlsinx_3cosx_3x_4^2 - 6Fcosx_3}{l(4(M+m) - 3mcos^2x_3)}
\end{aligned}$$

Al igual que en el caso anterior la ecuación linealizada es:

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{3mg}{4(M+m)} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{6(M+m)g}{4l(M+m)} & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ \frac{1}{(M+m)} \\ 0 \\ \frac{-3}{2l(M+m)} \end{bmatrix} u \tag{2.15}$$

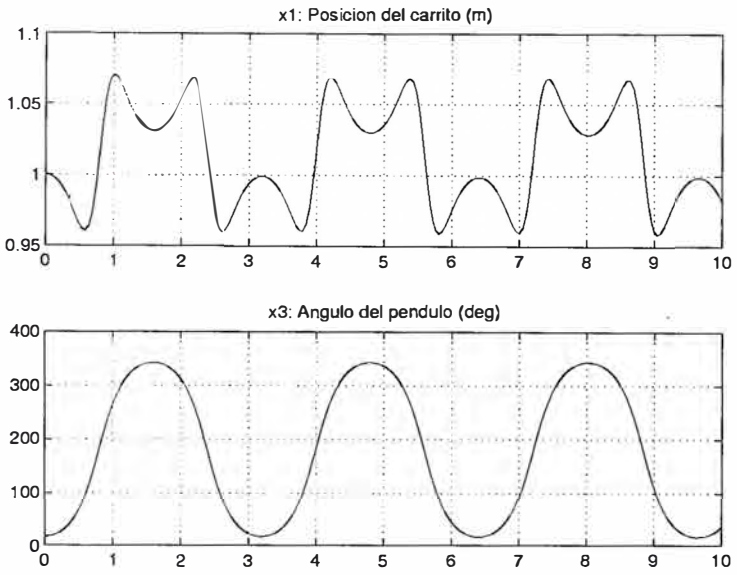

Figura 2.5: Simulación para el sistema PIBF. Caso 1

2.1.3 Simulaciones

Se simula el sistema PIBF con el próposito de verificar el modelo matemático. Las ecuaciones 2.11 que modelan el sistema PIBF son resueltas en el intervalo de tiempo $[0, 12]$ con los parámetros $M = 2\ Kg$, $m = 0.2\ Kg$ *y* $l = 0.6\ m$. Para resolver las ecuaciones se utiliza la función de integración provista por MATLAB: ode45. Se consideran dos casos:

- Con entrada nula (respuesta débido a condiciones iniciales).
- Con entrada diferente de cero y condiciones iniciales nulas (respuesta débida solo a la entrada).

Resultados

Se simuló el sistema PIBF considerando las siguientes condiciones:

1. $u = 0N$, no se le aplica fuerza externa al sistema y $x(0) = [1\ m, 0\ m/s, 20°, 0°/s]^T$. Los resultados se muestran en las figuras 2.5 y 2.6
2. $u = 2\ N$ y $x(0) = [0\ m, 0\ m/s, 0°, 0°/s]^T$. Los resultados se muestran en la figura 2.7

Observaciones y Conclusiones

1. Cuando no se aplica fuerza externa

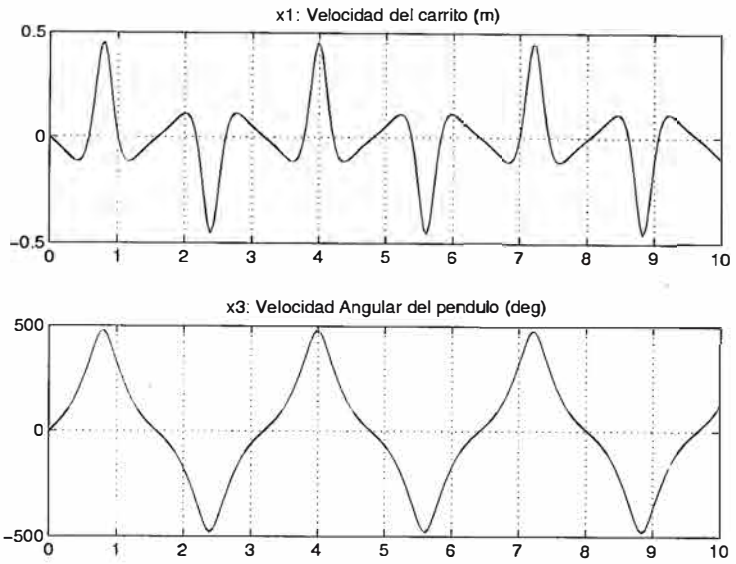

Figura 2.6: Simulación para el sistema PIBF. Caso 1

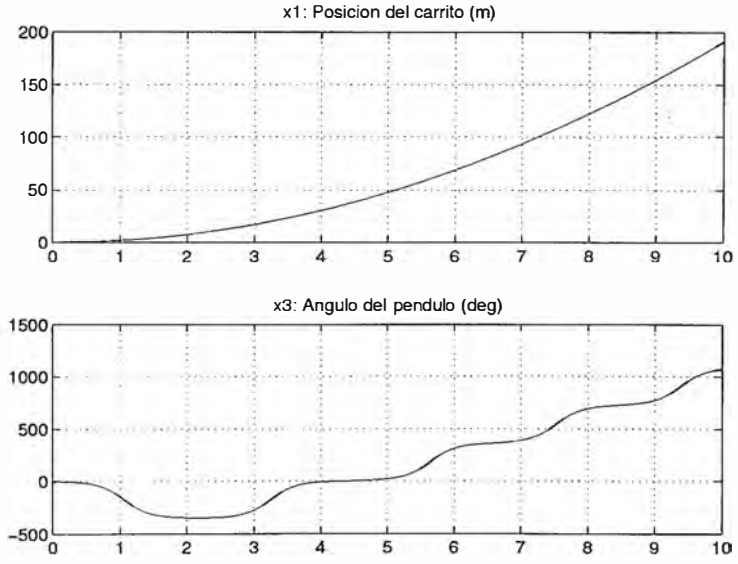

Figura 2.7: Simulación para el sistema PIBF. Caso 2

- El carrito oscila alrededor del punto $x_1 = 1\ m$ con un movimiento que se debe solamente a la oscilación del péndulo, pues la entrada u y la velocidad inicial $x_2(0)$ del carrito son cero.
- El ángulo $x_3(t)$ oscila entre $\pm 15°$ indefinidamente. Está oscilación se debe a la energía potencial inicial de $x_3(0)$ y a la no existencia de disipación (no hay fricción) de energía.
- Si $x_3(0) = 0$, la energía inicial sería nula y el sistema permaneceria en reposo en el estado $[1, 0, 0, 0]^T$.
- El periodo de oscilación es $3.1s$ aproximadamente.

2. Cuando se aplica una fuerza externa

   - Debido a la aceleración producida por la fuerza de entrada, el péndulo comienza a oscilar.
   - Esta oscilación se manifiesta en un efecto solitario apenas perceptible en la posición del carrito.
   - El movimiento del carrito es principalmente debido a la fuerza de entrada constante que produce una aceleración constante y un aumento cuadrático de la posición del carrito ($x_1(t)$) respecto al tiempo.
   - En este caso si existe un incremento de energía en el sistema, debido a la fuerza de entrada.
   - El péndulo comienza a girar indefinidamente en sentido horario, después de una sola oscilación. Esto significa que ha habido un paso de energía al movimiento giratorio.

Para el caso del sistema PIVF se realizarón las simulaciones correspondientes bajo las mismas condiciones del sistema anterior y con las mismas condiciones iniciales registrandose resultados idénticos que se muestran a continuación en las figuras 2.8, 2.9 y 2.10

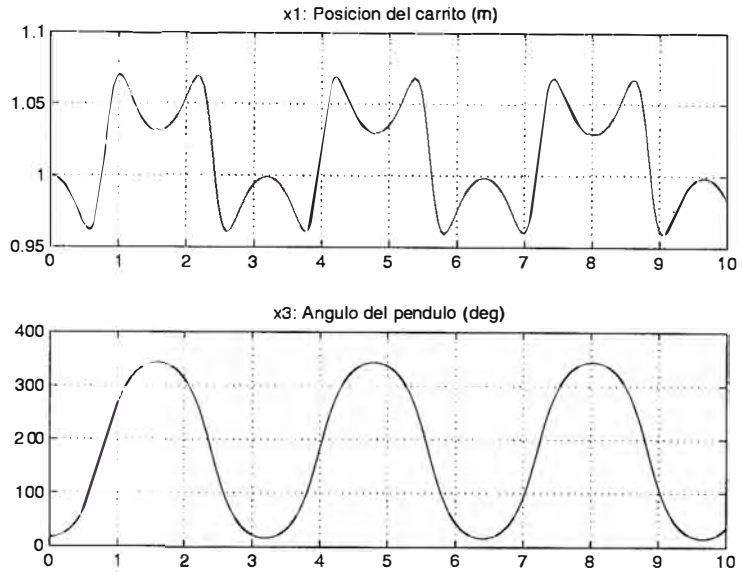

Figura 2.8: Simulación para el sistema PIVF. Caso 1

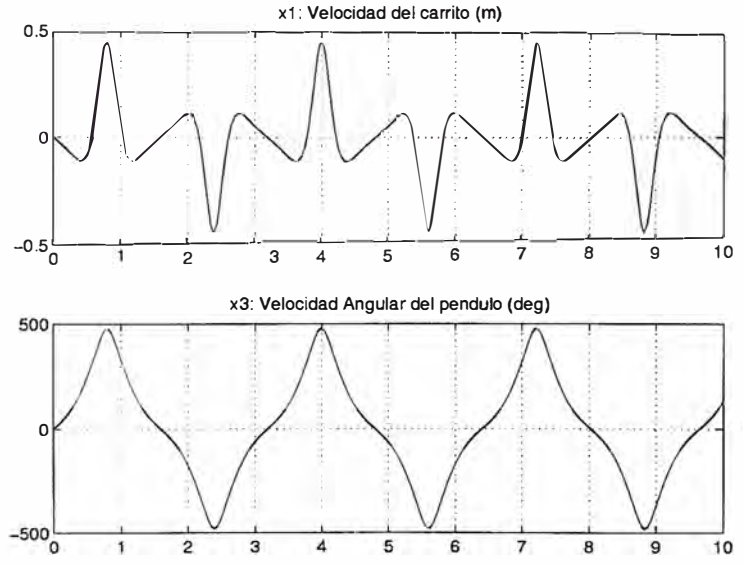

Figura 2.9: Simulación para el sistema PIVF. Caso 1

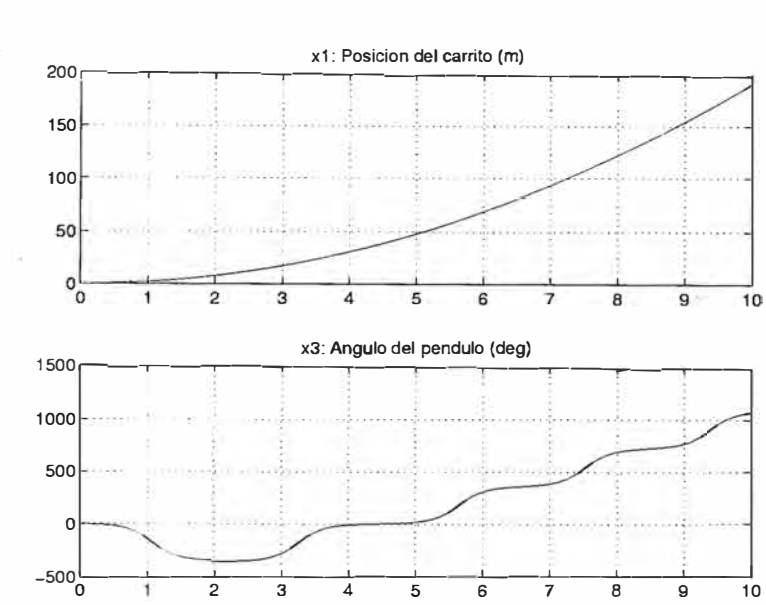

Figura 2.10: Simulación para el sistema PIVF. Caso 2

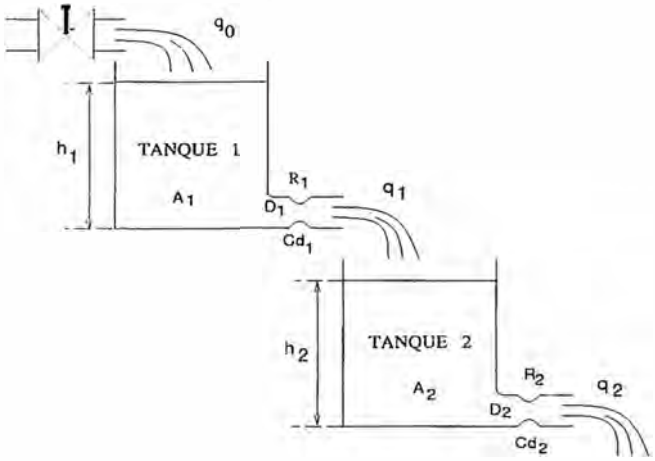

Figura 2.11: Tanques No Interconectados

2.2 Tanque Doble

Los procesos a modelar son dos tanques con líquido que pueden estar o no interconectados y que tienen como única entrada el caudal de ingreso y como salidas los niveles de líquido en los mismos tanques, tal como se muestran en las figuras 2.11 y 2.12.

2.2.1 Conceptos Teóricos

El flujo de fluidos en tanques y tuberías puede ser de tipo láminar o turbulento. El número de Reynolds (Re) determina esta característica según el valor que tome y está

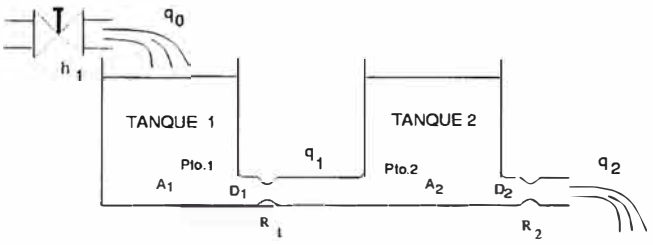

Figura 2.12: Tanques Interconectados

definido por la siguiente ecuación:

$$R_e = \frac{4Q}{\pi D \nu}$$

donde:

Q : Caudal que fluye por la tubería

D : diámetro de la sección recta de la tubería.

ν : viscosidad cinemática del fluido.

La viscosidad cinemática del agua a presión atmosférica y a temperatura ambiente es $0.975x10^{-6}\frac{m^2}{s}$. Si $R_e < 2000$, el flujo es láminar y las ecuaciones diferenciales son lineales. Si $R_e > 3000$, el flujo es turbulento y las ecuaciones diferenciales que gobiernan el sistema son no lineales. Cabe indicar que en los procesos industriales los flujos de fluidos en tuberías y tanques son frecuentemente turbulentos.

Resistencia Hidráulica

Se define la resistencia hidráulica como la razón entre la variación de diferencia de nivel y la variación en caudal.

$$R = \frac{dH}{dQ}$$

Balance de Energía

Para un flujo permanente e incompresible (ρ = constante) la ecuación de Bernoulli se expresa:

$$\frac{P}{\rho} + gz + \alpha\frac{V^2}{2} = constante$$

donde :

$P \rightarrow$ presión estática.

$z \rightarrow$ Altura respecto de un nivel de referencia.

$V \rightarrow$ velocidad de flujo.

$g \rightarrow$ constante de gravedad $(9.81\frac{m}{s^2})$.

$\alpha \rightarrow$ coeficiente de Coriolis.

El coeficiente de Coriolis es un factor de corrección debido a que el teorema de Bernoulli [16] fue establecido para una linea de corriente. Este coeficiente es cercano a 1 y tipicamente es 0.75. La ecuación de Balance de Energía se presenta como:

$$\frac{P_1}{\rho} + gz_1 + \frac{V_1^2}{2} = \frac{P_2}{\rho} + gz_2 + \frac{V_2^2}{2}$$

Pudo haberse considerado también la presión manométrica ($P_{man} = 0$) en lugar de la presión atmosférica, pero siempre que se haga el cambio en ambos miembros.

Teorema de Torricelli

La relación conocida como el Teorema de Torricelli [16] es :

$$V_2 = \sqrt{2gH}$$

Considerando condiciones reales la velocidad del fluido es afectada por las características físicas de la tubería lo que nos lleva a considerar un factor de descarga C_d dado por:

$$C_d = C_v C_c$$

donde C_d es el coeficiente de velocidad que toma en cuenta la fricción del paso del fluido por la tubería. Este coeficiente depende del tamaño y forma del orificio, asi como de la elevación de la superficie libre. El valor de C_v no es menor de 0.98 para orificios de forma fluido-dinámicas. Para orificios de forma no fluido-dinámicas, el chorro sufre una contracción al abandonar el depósito. El coeficiente de contracción C_c depende de la forma y tamaño del orificio, asi como de la elevación libre sobre el chorro, varía normalmente entre 0.60 para orificios de bordes agudos y 1 para orificios de perfiles fluido-dinámicos. Por ello la velocidad del flujo queda afectada como:

$$V_2 = (C_d\sqrt{2g})\sqrt{H} \tag{2.16}$$

Conservación de la Masa

Para el caso de un tanque de sección recta A con caudal de entrada q_{ent} y caudal de salida q_{sal}, tenemos que tratándose de un flujo incompresible, la densidad de éste es constante, por lo tanto se cumple que el caudal de entrada menos el caudal de salida durante el pequeño intervalo de tiempo dt es igual a la cantidad de líquido acumulada en el tanque. Luego:

$$(q_{ent} - q_{sal})dt = Adh \tag{2.17}$$

2.2.2 Modelo Matemático de un Tanque

El objetivo es encontrar la relación existente entre el caudal de entrada Q y la altura en el tanque H, tanto para flujo láminar como para flujo turbulento.

Flujo Laminar

Si el flujo es laminar tenemos la relación lineal entre el caudal de régimen y la carga hidrostática de régimen está dada por la siguiente relación:

$$Q = KH$$

donde K es una constante de dimensiones $[L^2T^{-1}]$. De la definición de resistencia hidráulica resulta:

$$R = \frac{dH}{dQ} = \frac{H}{Q} = \frac{1}{K}$$

Flujo Turbulento

Para este caso tenemos que la relación entre Q y H es no-lineal, y viene dado por:

$$Q = K\sqrt{H}$$

Está relación se obtiene de la ecuación de Torricelli donde la velocidad del fluido es $V \quad \sqrt{2gH}$. Para obtener el caudal multiplicamos la velocidad del fluido por el área de la sección de la tubería, obteniendo:

$$Q = SV = SC_d\sqrt{2g}\sqrt{H} = K\sqrt{H}$$

donde:

$Q \rightarrow$ caudal que fluye por la tubería

$H \rightarrow$ nivel del líquido en el tanque

$K \rightarrow$ constante.

De la definición de resistencia hidráulica tenemos:

$$R = \frac{dH}{dQ} = \frac{2H}{Q}$$

2.2.3 Tanques No Interconectados

Si el flujo es turbulento (es el más común en procesos industriales) partimos de la ecuación de Torricelli de donde la velocidad del fluido es $V = \sqrt{2gH}$. Para el caudal multiplicamos la velocidad del fluido por el área de la sección de la tubería:

$$Q = SV = SC_d\sqrt{2g}\sqrt{H} = K\sqrt{H}$$

donde:

$C_d \rightarrow$ factor de descarga

$S \rightarrow$ área de la sección de la tuberia

$Q \rightarrow$ caudal que fluye por la tubería

$H \rightarrow$ nivel del líquido en el tanque

$K \rightarrow$ constante.

Luego la resistencia hidráulica es:

$$R = \frac{dH}{dQ} = \frac{2\sqrt{H}}{K}$$

Para flujo láminar tenemos una relación lineal entre el caudal de régimen y la carga hidrostática de régimen:

$$Q = KH$$

Luego la resistencia hidráulica:

$$R = \frac{dH}{dQ} = \frac{H}{Q} = \frac{1}{K}$$

El modelo no lineal del sistema de dos Tanques No Interconectados (2.11) para régimen turbulento se obtiene aplicando el teorema de Torricelli y el principio de conservación de la masa:

$$q_1 = K_1\sqrt{h_1}$$

donde

$$K_1 = S_1\sqrt{2g}C_{d1}$$

$$(q_0 - q_1)dt = A_1 dh_1$$

y usando la definición de resistencia hidráulica, tenemos para el tanque 1:

$$(q_0 - q_1)dt = A_1 R_1 dq_1$$

$$(q_0 - q_1)dt = A_1 \frac{2q_1}{K_1^2} dq_1$$

que resulta en:

$$\dot{q_1} = (q_0 \frac{K_1^2}{2A_1})\frac{1}{q_1} - \frac{K_1^2}{2A_1}$$

y en función de la altura h_1:

$$\dot{h_1} = \frac{q_0}{A_1} - \frac{K_1}{A_1}\sqrt{h_1}$$

De igual forma procedemos para el tanque 2:

$$\dot{q_2} = (q_1 \frac{K_2^2}{2A_2})\frac{1}{q_2} - \frac{K_2^2}{2A_2}$$

y en términos de la altura $h2$:

$$\dot{h_2} = \frac{K_1}{A_2}\sqrt{h_1} - \frac{K_2}{A_2}\sqrt{h_2}$$

obteniendo las siguiente ecuación de estado no lineal para régimen turbulento:

$$\dot{h} = \begin{bmatrix} \dot{h_1} \\ \dot{h_2} \end{bmatrix} = \begin{bmatrix} \frac{q_0}{A_1} - \frac{K_1}{A_1}\sqrt{h_1} \\ \frac{K_1}{A_2}\sqrt{h_1} - \frac{K_2}{A_2}\sqrt{h_2} \end{bmatrix} \tag{2.18}$$

donde:

$$K_1 = S_1\sqrt{2g} * C_{d_1}$$

$$K_2 = S_2\sqrt{2g} * C_{d_2}$$

Esta ecuación se puede expresar como función de h y q_0:

$$\dot{h} = f(h, q_0) = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

El modelo linealizado se obtiene a partir de la ecuación (2.18) usando expansión en series de Taylor alrededor del punto de equilibrio o de régimen permanente. La justificación de usar Taylor para la linealización se basa en que las variaciones alrededor del punto de equilibrio sean lo suficientemente pequeñas como para despreciar los términos que contengan derivadas de orden superior a 1. Considerando $\overline{h_1}, \overline{h_2}$ el punto de equilibrio, se tiene las condiciones de equilibrio:

$$\overline{q_0} = K_1\sqrt{\overline{h_1}}$$

$$\overline{h_1} = \left(\frac{K_2}{K_1}\right)^2 \overline{h_2}$$

Linealizando en la vecindad de $\overline{h} = \begin{bmatrix} \overline{h_1} \\ \overline{h_2} \end{bmatrix}$:

$$\begin{bmatrix} \dot{h_1} \\ \dot{h_2} \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}|_{(\overline{h_1},\overline{q_0})} + \begin{bmatrix} \frac{\partial f_1}{\partial h_1} & \frac{\partial f_1}{\partial h_2} \\ \frac{\partial f_2}{\partial h_1} & \frac{\partial f_2}{\partial h_2} \end{bmatrix}|_{(\overline{h_1},\overline{q_0})} \begin{bmatrix} h_1 - \overline{h_1} \\ h_2 - \overline{h_2} \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1}{\partial q_0} \\ \frac{\partial f_2}{\partial q_0} \end{bmatrix}|_{(\overline{h_1},\overline{q_0})} (q_0 - \overline{q_0}) \tag{2.19}$$

Reemplazando la ecuación 2.18 en 2.19 se tiene:

$$\dot{h} = \dot{\overline{h}} + \begin{bmatrix} -\frac{K_1}{2A_1\sqrt{\overline{h_1}}} & 0 \\ \frac{K_1}{2A_2\sqrt{\overline{h_1}}} & -\frac{K_2}{2A_2\sqrt{\overline{h_2}}} \end{bmatrix} \Delta h + \begin{bmatrix} \frac{1}{A_1} \\ 0 \end{bmatrix} \Delta q_0 \tag{2.20}$$

donde:

$$\dot{h} = \begin{bmatrix} \dot{h_1} \\ \dot{h_2} \end{bmatrix}$$

$$\dot{\overline{h}} = \begin{bmatrix} \frac{\overline{q_0}}{A_1} - \frac{K_1}{A_1}\sqrt{\overline{h_1}} \\ \frac{K_1}{A_2}\sqrt{\overline{h_1}} - \frac{K_2}{A_2}\sqrt{\overline{h_2}} \end{bmatrix}$$

$$\Delta h = \begin{bmatrix} h_1 - \overline{h_1} \\ h_2 - \overline{h_2} \end{bmatrix}$$

$$\Delta q_0 = q_0 - \overline{q_0}$$

La matriz $\overline{h}$ nos da la solución nominal del sistema linealizado. Teniendo en cuenta que en régimen permanente $\overline{h}$ es constante:

$$\dot{\overline{h}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

De donde resultan dos condiciones de equilibrio:

$$\overline{q_0} = K_1\sqrt{\overline{h_1}}$$

$$\overline{h_1} = (\frac{K_2}{K_1})^2\overline{h_2}$$

La ecuación 2.20 se puede expresar en la forma de ecuación de estado:

$$\dot{\Delta h} = \begin{bmatrix} -\frac{1}{A_1 R_1} & 0 \\ \frac{1}{A_2 R_1} & -\frac{1}{A_2 R_2} \end{bmatrix} \Delta h + \begin{bmatrix} \frac{1}{A_1} \\ 0 \end{bmatrix} \Delta q_0 \qquad (2.21)$$

donde los número de Reynolds (Re) son:

$$R_1 = \frac{2\sqrt{\overline{h_1}}}{K_1}$$

$$R_2 = \frac{2\sqrt{\overline{h_2}}}{K_2}$$

2.2.4 Tanques Interconectados

Para el caso de Tanques Interconectados (figura 2.11) el objetivo es modelar el sistema en función del caudal de entrada (q_0) y las alturas del nivel de líquido en los tanques 1 y 2 (h_1 y h_2), además de encontrar las expresiones para los caudales en las tuberías de área de sección S_1 y S_2. Para este caso la velocidad del fluido por la tubería que une los tanques 1 y 2 es $V = \sqrt{2g(h_1 - h_2)}$, considerando que h_1 siempre sea mayor que h_2, pues nuestro modelo está dado para esta condición. El caudal es expresado como:

$$q_1 = S_1 C d_1 \sqrt{2g}\sqrt{h_1 - h_2}$$

considerando S_1 el área de la tubería y C_{d1} es el coeficiente de descarga. Para el tanque 1 aplicando ec. 2.17 tenemos:

$$(q_0 - q_1) = A_1 \dot{h_1}$$

y para el tanque 2 aplicando ecs. 2.16 y 2.17 tenemos:

$$q_2 = S_2 C d_2 \sqrt{2g}\sqrt{h_2}$$

$$(q_1 - q_2 = A_2 \dot{h_2}$$

Usando las ecuaciones del primer tanque tenemos la primera ecuación diferencial:

$$\dot{h_1} = \frac{q_0}{A_1} - \frac{K_1}{A_1}\sqrt{h_1 - h_2}$$

Y de las ecuaciones para el segundo tanque tenemos:

$$\dot{h_2} = \frac{K_1}{A_2}\sqrt{h_1 - h_2} - \frac{K_2}{A_2}\sqrt{h_2}$$

Luego la ecuación vectorial $\dot{h} = f(h, q_0)$ no lineal del sistema es:

$$\dot{h} = \begin{bmatrix} \dot{h_1} \\ \dot{h_2} \end{bmatrix} = \begin{bmatrix} \frac{q_0}{A_1} - \frac{K_1}{A_1}\sqrt{h_1 - h_2} \\ \frac{K_1}{A_2}\sqrt{h_1 - h_2} - \frac{K_2}{A_2}\sqrt{h_2} \end{bmatrix} \tag{2.22}$$

donde:

$$K_1 = S_1\sqrt{2g} * Cd_1$$

$$K_2 = S_2\sqrt{2g} * Cd_2$$

Los puntos de equilibrio para una entrada estacionaria $\overline{q_0}$ son:

$$\overline{h_2} = \frac{\overline{q_0}^2}{K_2^2}$$

$$\overline{h_1} = \overline{q_0}^2\left(\frac{1}{K_1^2} + \frac{1}{K_2^2}\right)$$

Se considerará siempre que $h_1 > h_2$. Las ecuaciones del modelo linealizado finalmente son:

$$\dot{\Delta h} = \begin{bmatrix} -\frac{1}{A_1 R_1} & \frac{1}{A_1 R_1} \\ \frac{1}{A_1 R_1} & -\frac{1}{A_2}\left(\frac{1}{R_1} + \frac{1}{R_2}\right) \end{bmatrix} \Delta h + \begin{bmatrix} \frac{1}{A_1} \\ 0 \end{bmatrix} \Delta q_0 \tag{2.23}$$

donde:

$$R_1 = \frac{2\sqrt{\overline{h_1} - \overline{h_2}}}{K_1}$$

$$R_2 = \frac{2\sqrt{\overline{h_2}}}{K_2}$$

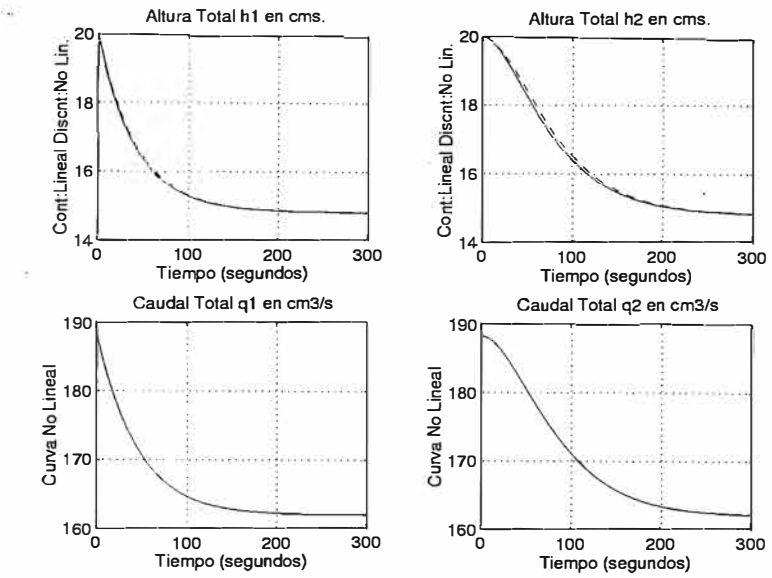

Figura 2.13: Simulación para los Tanques No Interconectados

2.2.5 Simulación

Para comprobar que los modelos linealizados se aproximan a los modelos reales (no-lineales) para valores cercanos al punto de equilibrio, y asegurar que se está trabajando efectivamente en régimen de flujo turbulento (número de Reynolds mayores a 3000), se efectuo la simulación de ambos casos usando MATLAB. Suponiendo los siguientes parámetros de planta:

- Area de los tanques: $A_1 = A_2 = 225\ cm^2$
- Diámetro de la tubería de los tanques $D_1 = D_2 = 1.27\ cm$
- Coeficientes de descarga: $C_{d1} = C_{d2} = 0.75$

Para el caso de Tanques No Interconectados se tiene que las alturas de los tanques en régimen nominal es $h_1 = h_2 = 15\ cm$ y la altura de los tanques de 25 cm o más. Asimismo el caudal en régimen nominal $q_0 = 162.9873\ \frac{cm^3}{s} = 155\ \frac{galUS}{hora}$. Las ecuaciones diferenciales a resolver se dan por las ecs. 2.18 (solución no lineal) y 2.21 (solución lineal para las alturas) Los resultados se muestran en la figura 2.13 para $h_{1inicial} = h_{2inicial} = 20\ cm$. De aquí se concluye:

- Cuando las alturas iniciales de ambos tanques están en el intervalo de [10, 20] cm. no hay prácticamente error de linealidad. Esto viendo que las curvas del modelo linealizado se aproximan bastante bien a las curvas correspondientes al modelo no-lineal.

- Se comprueba que el flujo en las tuberias de salida de ambos tanques es turbulento, justificandose el modelo matemático usado. De la figura **??** se tiene que el número de Reynolds es mayor a 3000 en todo instante.

- El tiempo de asentamiento para el tanque 1 (T_{s1}) está dado por la relación:

$$T_{s1} = -A_1 R_1 Ln\left(\frac{0.005\overline{h_1}}{h_1(0) - \overline{h_1}}\right)$$

donde:

$$\begin{aligned} R_1 &\rightarrow \text{Resistencia hidráulica.} \\ h_1(0) &\rightarrow \text{Altura inicial en el tanque 1.} \end{aligned}$$

Para el tanque 2 el tiempo de asentamiento (T_{s2}) está dado por la relación:

$$e^{-\frac{T_{s2}}{A_1 R_1}}\left[\frac{T_{s2}}{A_2 R_1}(h_1(0) = \overline{h_1}) + (h_2(0) - \overline{h_2})\right] = 0.005\overline{h_2}$$

Evaluando las expresiones anteriores con los parámetros elegidos, considerando una altura inicial $h_1(0) = h_2(0) = 20\ cm$ resulta $T_{s1} = 173.9\ s.$ y $T_{s2} = 255\ s.$. De las relaciones anteriores vemos que el tiempo de asentamiento para ambos tanques es mayor si el área de los tanques aumenta y/o el diámetro de las tuberías disminuye.

Para el caso de Tanques Interconectados se asumio que el modelo no lineal se construyó asumiendo flujo turbulento en las tuberías, y el caudal en cualquiera de las tuberias mayor a 20 $\frac{cm^3}{seg}$ para cumplir la condición. Aquí se fijaron los siguientes valores:

- Caudal de régimen nominal $\overline{q_0} = 150\ \frac{cm^3}{s}$

- Altura del tanque 1 en régimen nominal $h_1 = 25.4\ cm.$

- Altura del tanque 2 en régimen nominal $h_2 = 12.7\ cm.$

De la simulación se concluye:

- Para variaciones alrededor del punto de reposo de las alturas en un rango de $\pm 10\ cm$ no se registran caudales menores de 20 $\frac{m^3}{seg}$, condición para que exista flujo turbulento.

- Las variaciones que se hacen en el rango de $\pm 10\ cm$ alrededor de los valores nominales de las alturas en los tanques 1 y 2 deen cumplir la condición $h_1 > h_2$ para no generar indeterminación en la simulación.

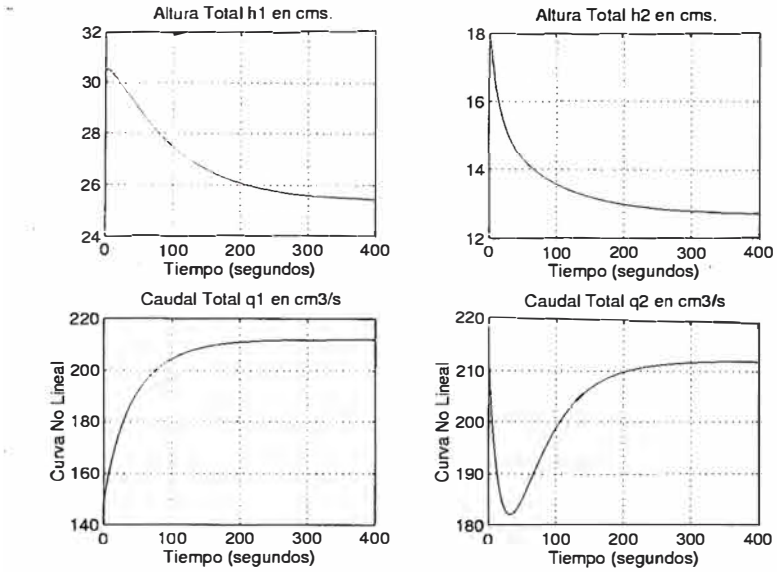

Figura 2.14: Simulación para los Tanques Interconectados

- Dentro del rango mencionado anteriormente la aproximación lineal al modelo no lineal es satisfactorio.

- El tiempo de asentamiento de las alturas 1 y 2 aumenta si la variación de las alturas es hacia un mismo sentido, es decir si ambas son positivas o negativas, de lo contario el tiempo de asentamiento disminuirá. En las gráficas de la figura 2.14 vemos los resultados para $\Delta h_1 = +5\ cm.$ y $\Delta h_2 = +5\ cm.$

CAPÍTULO III
CONTROL DE PROCESOS

3.1 Diseño de Controladores

3.1.1 Controlador Realimentado de Estado

Se ha seleccionado el *Controlador Realimentado de Estado* para obtener el comportamiento deseado ubicando los polos del sistema de lazo cerrado en el semiplano izquierdo de s, la variable de la transformada de Laplace.

Está técnica de control sólo es aplicable a sistemas lineales que sean completamente controlables [16]. Las plantas utilizadas son no lineales por lo tanto el control realimentado de estado no es directamente aplicable. Debido a que se trata de un problema de regulación, es posible considerar que las plantas operan en una pequeña región centrada en el punto de operación. Luego los modelos de las plantas se pueden linealizar alrededor de su punto de operación y obtener un **modelo linelizado** que se emplea para diseñar el controlador. A continuación se describe el procedimiento de diseño. Sea el sistema definido por la ecuación de estado

$$\dot{x} = Ax + Bu \tag{3.1}$$

donde:

$x \rightarrow$ vector de estado de dimensión n

$u \rightarrow$ señal de control

$A \rightarrow$ matriz de dimensión nxn y

$B \rightarrow$ matriz de dimensión $nx1$.

Luego la señal de control u realimentada de estado tiene la forma general:

$$u = -Kx + r \tag{3.2}$$

donde:

$K \rightarrow$ matriz de realimentación, 1xn

$r \rightarrow$ señal de referencia.

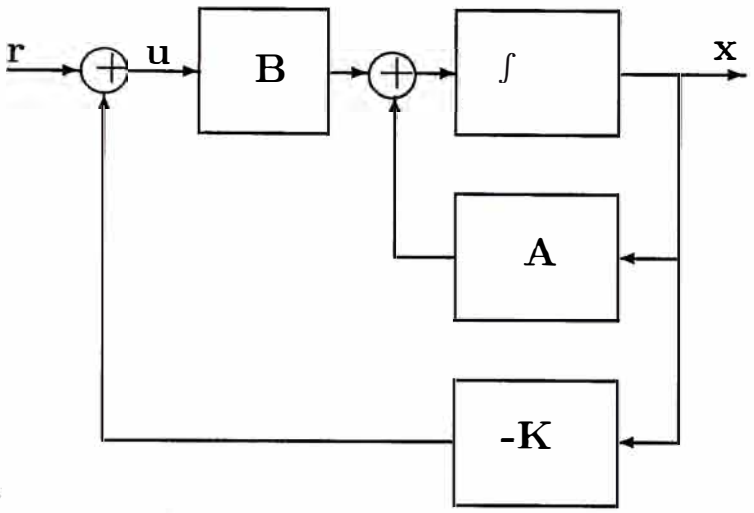

Figura 3.1: Diagrama de Bloques del Sistema Realimentado de Estado

Para el péndulo invertido $r = 0$ y el doble tanque $r = h_0$ la altura nominal. La figura 3.1.1 muestra un diagrama de bloques de la planta-controlador. De las ecuaciones (3.1) y (3.2) resulta la siguiente ecuación de lazo cerrado:

$$\dot{x} = (A - BK)x + Br \tag{3.3}$$

Luego los polos del sistema de lazo cerrado estan determinados por los eigenvalores de la matriz $(A - BK)$. Si el sistema es completamente controlable es posible obtener la matriz K tal que los eigenvalores de $A - BK$ se ubiquen en el lugar deseado. Existen varios procedimientos para calcular K entre ellos tenemos la fórmula de Ackerman [16]. A continuación se presentan el diseño del controlador y simulación del controlador-planta para el caso del péndulo invertido.

Diseño y Simulación : Péndulo Invertido

Primero se verifica que el sistema efectivamente tenga controlabilidad de estado completa, [16] y controlabilidad de salida, como se puede demostrar de los archivos de simulación de MATLAB. El sistema de control que se simula se describe en la ecuación (??):

$$\dot{x} = f(x, u)$$

donde:

$$u = -Kx.$$

Luego

$$\dot{x} = f(x, -Kx).$$

$M = 2$	$m = 0.2$	$l = 0.6$	$g = 9.8$

Tabla 3.1: Parámetros del Sistema PI

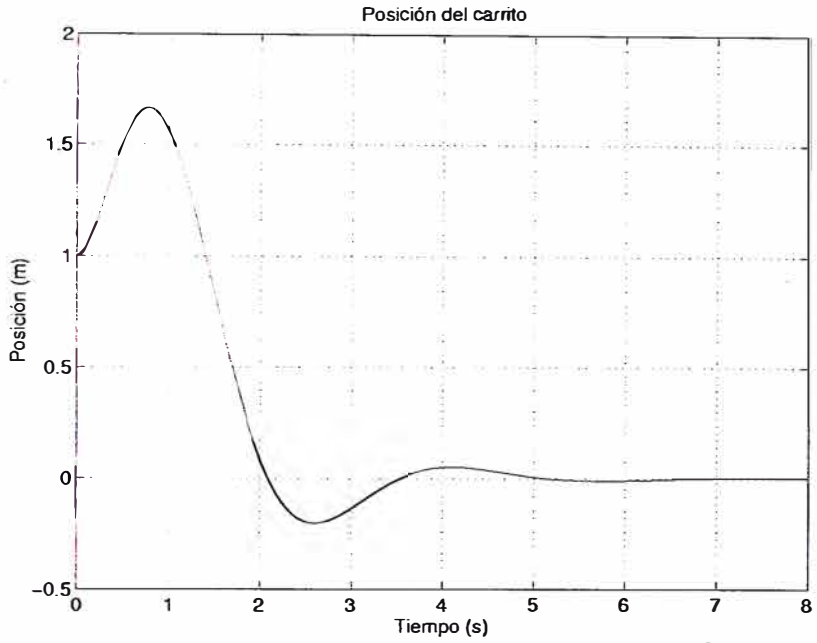

Figura 3.2: Posición del carrito en metros

Los parámetros del sistema, considerados para la simulación son los indicados en la Tabla 3.1.1. La ganancia K del controlador es:

$$K = \begin{bmatrix} -3.6735 & -4.5306 & -48.9641 & -11.1184 \end{bmatrix}$$

la cual ha sido calculada para que los polos de lazo cerrado del sistema linealizado sean $[-1-2i,\ -1+2i,\ -3,\ -4]$. El tiempo de simulación considerado es de diez segundos y las condiciones iniciales son $x(0) = [1,\ 0,\ 0.3,\ 0]$ (unidades MKS).

Los resultados de las simulaciones son mostrados en las figuras 3.2, 3.3 y 3.4. De estas figuras se puede apreciar que el Péndulo Invertido es controlado satisfactoriamente en un tiempo de 6 segundos aproximadamente. Este tiempo se puede modificar, cambiando el valor de K.

Para el caso del sistema PIVF tenemos que para las mismas condiciones iniciales, tiempo de simulación y considerando los mismos polos de lazo cerrado tenemos que la ganancia del controlador es:

$$K = \begin{bmatrix} -5.7820 & -5.6856 & -51.1528 & -10.1942 \end{bmatrix}$$

Los resultados de las simulaciones son mostrados en las figuras 3.6, 3.7. 3.8 y 3.9.

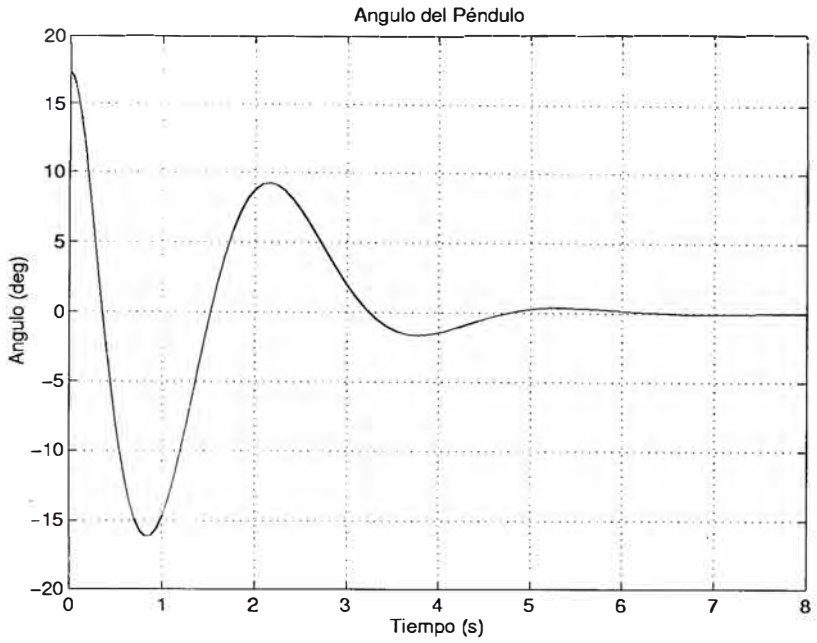

Figura 3.3: Angulo del Péndulo Invertido en grados

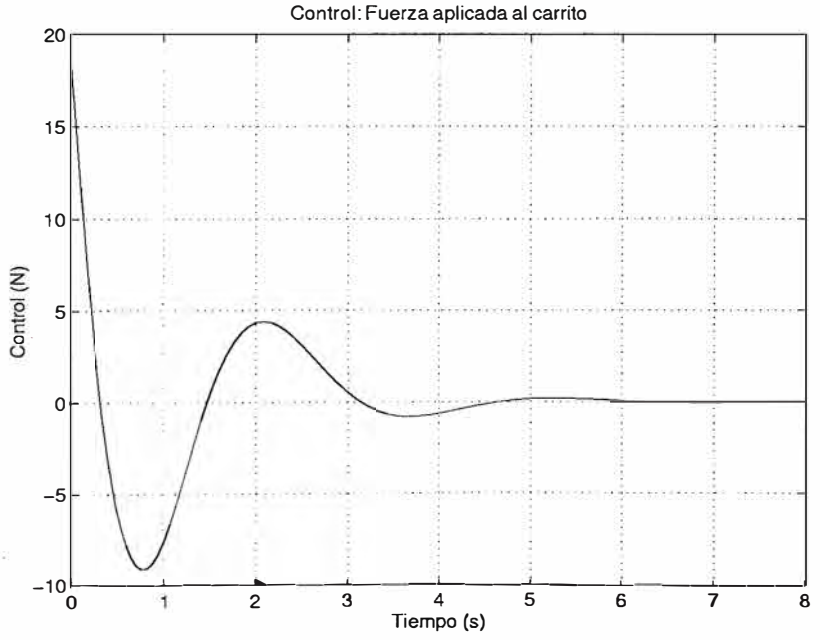

Figura 3.4: Control: Fuerza aplicada al carrito (en Newtons)

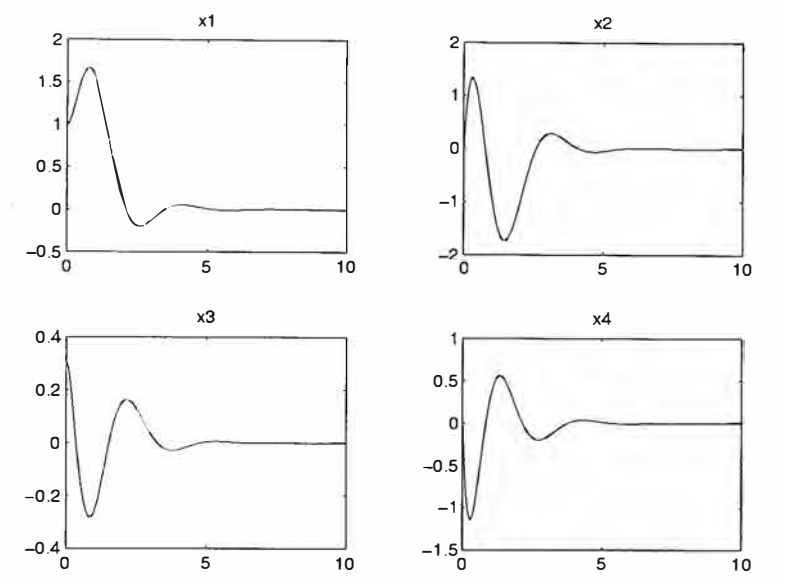

Figura 3.5: Simulación del sistema PIBF

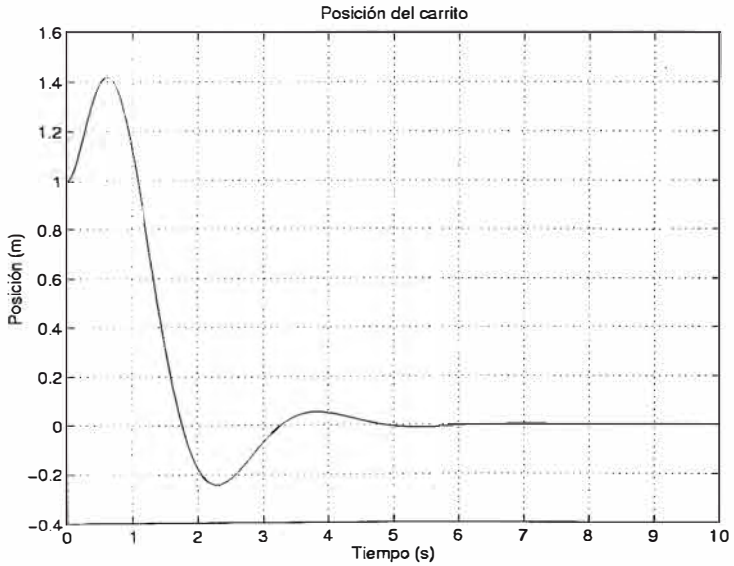

Figura 3.6: Posición del carrito en metros

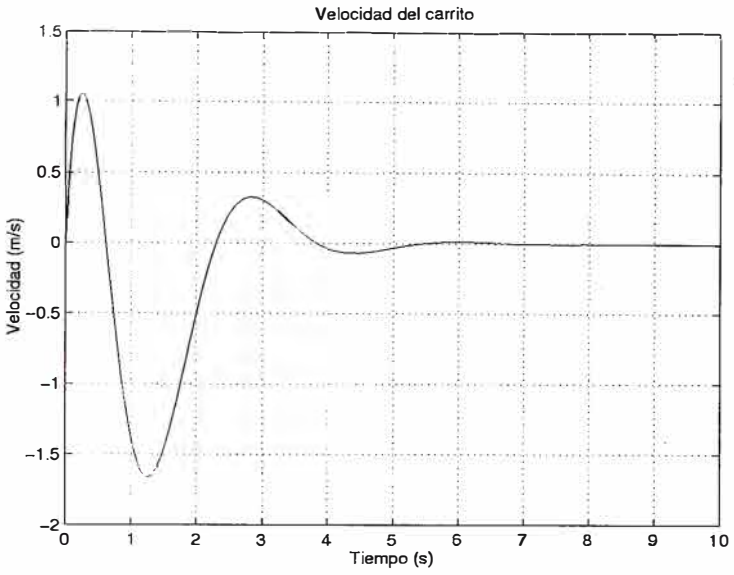

Figura 3.7: Velocidad del Péndulo Invertido

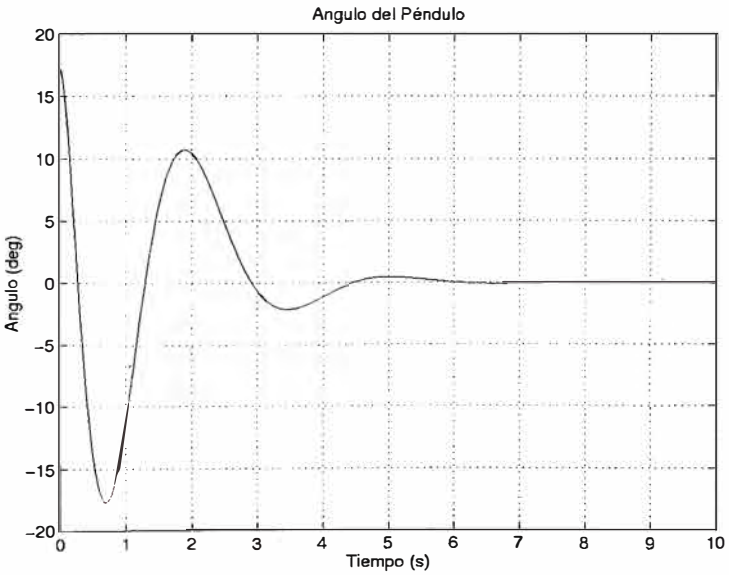

Figura 3.8: Angulo del Péndulo Invertido en grados

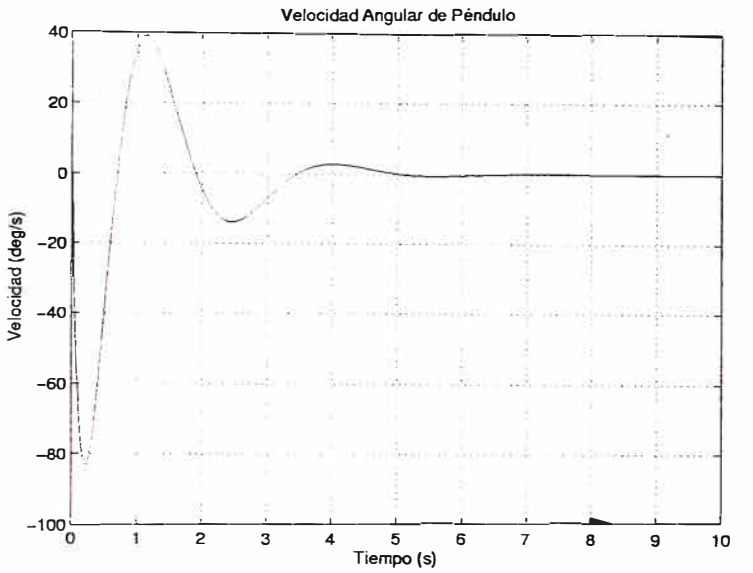

Figura 3.9: Velocidad angular del Péndulo Invertido

Diseño y Simulación : Tanque Doble

El sistema de control que se simula se describe en la ecuación (2.21). Primero se verifica que el sistema efectivamente tenga controlabilidad completa, [16] y controlabilidad de salida, como se puede demostrar de los archivos de simulación de MATLAB.

Entonces asumiendo que los polos de lazo cerrado $[-0.05, -0.05]$ resulta que la ganancia K del controlador es:

$$K = [11.6342 \quad 6.2284]$$

Asumiendo un rango de trabajo entre 10 y 20 cm. tanto para h_1 como para h_2. En la simulación se tiene para un caso extremo $h_{inic1} = h_{inic2} = 20\ cm$ un valor de caudal $q_0 = 70\ \frac{cm^3}{s}$, mientras que para el otro caso extremo $h_{inic1} = h_{inic2} = 10\ cm$ un valor de caudal $q_0 = 250\ \frac{cm^3}{s}$. Es decir, se requiere una bomba con capacidad de 300 $\frac{cm^3}{s}$, dando el margén de seguridad para que la bomba no trabaje nunca al máximo. El tiempo de asentamiento del sistema controlado es de alrededor de $T_s = 120\ seg.$. Finalmente, a medida que los polos están más alejados del origen, el tiempo de asentamiento disminuye pero el valor de la matriz K es mayor y en consecuencia la señal de control q_0 aumentaria en exceso, lo cual no estaria bien porque requeriría una bomba de mayor potencia.

El sistema de control de Tanques Interconectados que se simula se describe en la ecuación (2.23). Primero se verifica que el sistema efectivamente tenga controlabilidad completa y controlabilidad de salida, como se puede demostrar de los archivos de simulación de MATLAB. Entonces asumiendo que los polos de lazo cerrado $[-0.05, -0.05]$

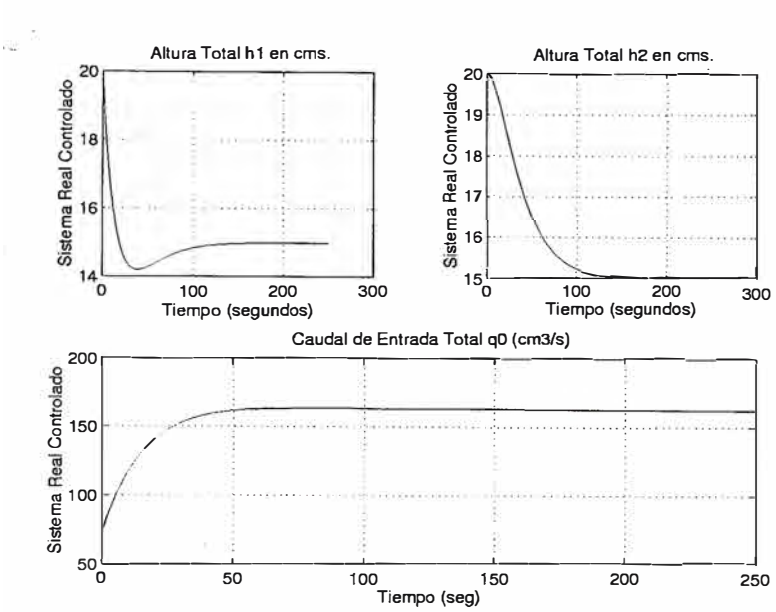

Figura 3.10: Simulación con controlador de sistema Tanque Doble. Caso Tanques No Interconectados

resulta que la ganancia K del controlador es:

$$K = [9.9749 \quad 2.0142]$$

De las simulaciones se encuentra un tiempo de establecimiento para el sistema controlado de aproximadamente $t_s = 350$ seg. Teniendo que $h_{inic1} = 30.5\ cm$ y $h_{inic2} = 18\ cm$ y un valor de caudal $q_0 = 250\ \frac{cm^3}{s}$. Se requiere entonces una bomba con capacidad de 250 $\frac{cm^3}{s}$, dando el margén de seguridad para que la bomba no trabaje nunca al máximo.

3.1.2 Controlador PID

El modelo matemático de un controlador PID está dado por:

$$H(s) = K_p + \frac{K_i}{s} + K_d s$$

Controlando el péndulo invertido con un controlador del tipo PID se tiene que la derivada de la señal de control es:

$$\dot{u} = K_d \ddot{e} + K_p \dot{e} + K_i e$$

Y la función de transferencia de la planta $G(s)$ es:

$$G(s) = c(sI - A)^{-1} b$$

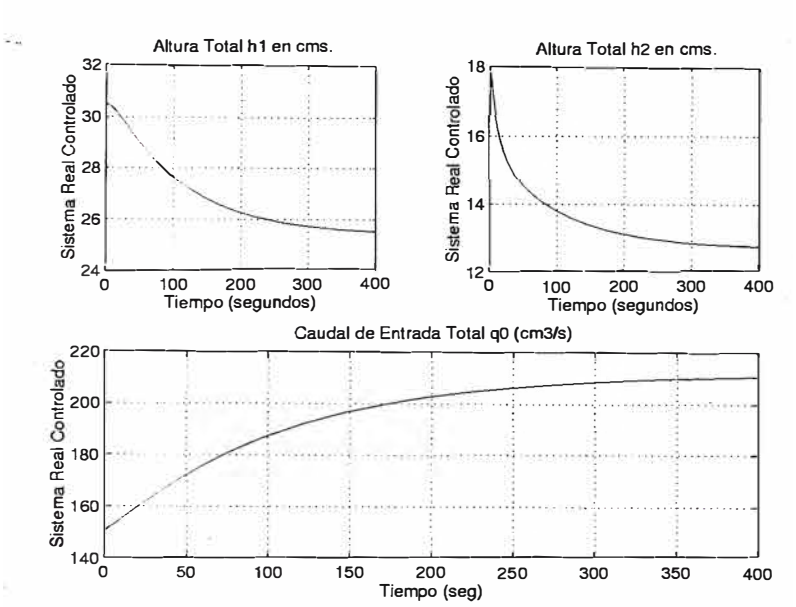

Figura 3.11: Simulación con controlador de sistema Tanque Doble. Caso Tanques Interconectados

Resolviendo la ecuación anterior utilizando el modelo linealizado del péndulo invertido de bola (pibf) encontrado en el capítulo anterior resulta que el polinomio caracteristico del sistema es: [1]

$$lM^2s^5 + K_dM(l+1)s^4 + (K_p(l+1) - g(M+m))Ms^3 + (K_iM(l+1) - gK_d(M+2m))s^2$$

$$-gK_p(M+2m)s - gK_i(M+2m) = 0$$

Utilizando el criterio de Routh-Hurwitz para analizar la estabilidad del sistema tenemos que para un controlador de tipo P ($K_d = K_i = 0$) el sistema nunca es estable pues el polinomio característico es incompleto. Para el caso de un controlador de tipo PD o tipo PI llegamos a la misma respuesta, el polinomio característico es incompleto. Llegamos a la conclusión entonces que no se puede estabilizar con este controlador PID y se recomienda usar el controlador realimentado de estado.

[1] Se utilizó paquete Mathematica para solución de ecuación

CAPÍTULO IV
SISTEMA DE SOFTWARE

4.1 El Plan de Proyecto de Software

El Plan de Proyecto de Software se produce como culminación de la etapa de planificación. Su fin es proporcionarnos una guia de base con información de costos y de agenda que se utilizara a lo largo del ciclo de vida del software. Este Plan de Proyecto debe cumplir con los siguientes objetivos:

1. Comunicar el ámbito y los recursos a los gestores del Software.
2. Definir los riesgos y sugerencia de técnicas de aversión al riesgo.
3. Definir el coste y la agenda de la revisión de la gestión del Proyecto.
4. Proporcionar un enfoque global del desarrollo del Software para toda la gente involucrada en el Proyecto.

Con el propósito de estimar el esfuerzo que se requerira para el Proyecto de Software se puede utilizar una serie de opciones disponibles para el planificador:

1. Técnicas de descomposición para generar estimaciones de los costes y esfuerzo del Proyecto.
2. Desarrollar un modelo empírico para el cálculo de costes y esfuerzo del Software.
3. Utilizar herramientas automáticas de estimación.

Las técnicas de descomposición utilizan un enfoque de "Divide y Venceras" para la estimación del Proyecto de Software. Si el problema a resolver es muy complicado, se necesita subdividir el problema hasta llegar a problemas más manejables; de tal manera que se puedan resolver esos últimos individualmente para finalmente obtener una solución total de la combinación de las soluciones obtenidas.

Los modelos empíricos de estimación son un complemento de las técnicas de descomposición y se basan en la experiencia del grupo de desarrollo del proyecto. Otros modelos de estimación para el software utilizan fórmulas derivadas empiricamente para predecir datos que se requieren en el paso de planificación del Proyecto de Software, como el esfuerzo (en personas/mes), la duración del proyecto (en meses cronológicos) y otros datos relativos al proyecto. Dentro de este contexto Basili [17] describe cuatro

clases de modelos de recursos: modelos univariables estáticos, modelos multivariables estáticos, modelos multivariables dinámicos y modelos teóricos. Un modelo univariable estático toma la forma:

$$Recurso = c_1 \ x \ (caracteristicaestimada)^{c_2}$$

donde el recurso podría ser el esfuerzo, la duración del proyecto, la cantidad de personal o las lineas requeridas de documentación de software. Las constantes c_1 y c_2 se derivan de los datos recopilados de anteriores proyectos. La característica estimada puede ser la cantidad de lineas de código fuente, el esfuerzo u otra característica del software. La versión básica del modelo de coste constructivo o COCOMO es un ejemplo de modelo univariable estático.

Los modelos multivariables estáticos usan los datos históricos para obtener relaciones empíricas. Un modelo típico de esta categoria toma la forma:

$$Recurso = c_{11}er + c_{21}e2 + ...$$

donde e_i es la característica i-ésima del software y c_{i1}, c_{i2} son constantes obtenidas empíricamente para la característica i-ésima.

Un modelo multivariable dinámico proyecta proyecta los requisitos de recursos como una función del tiempo Si se obtiene empíricamente el modelo, los recursos se definen en una serie de pasos consecutivos en el tiempo que asignan cierto procentaje de esfuerzo (o de otro recurso) a cada etapa del proceso de ingeniería del software.

El último tipo de modelo de recursos examina el software desde un punto de vista microscópico, es decir, las características del código fuente (p. ej.: número de operadores y de operandos).

Las herramientas automáticas de estimación implementan una o varias técnicas de descomposición ó modelos empíricos. En sistemas de este tipo, se describen las características del grupo de desarrollo (por ejemplo : la experiencia, el ambiente de trabajo), y el software a desarrollar. De estos datos se obtienen las estimaciones de coste y esfuerzo.

4.2 Plan de Proyecto de Software SCD_Avanzado

4.2.1 Objetivos del Proyecto

Objetivos

1. Utilizar procesos de interes práctico para los ámbitos de la enseñanza e investigación .

2. Desarrollar los modelos matemáticos y simulaciones de Sistemas de Control que empleen técnicas de control moderno.

3. Desarrollar un Sistema de Control Distribuido adecuado para la enseñanza, investigación y desarrollo. Para la enseñanza pues aprovechando las características del Sistema X Window se puede realizar el aprendizaje correspondiente con el profesor desarrollando los conocimientos necesarios en una computadora y los alumnos observando los procedimientos y resultados en las pantallas de sus terminales; ganandose con ello dinámica en las clases. Para la investigación pues agregar otros modelo de proceso a la aplicación se verá facilitado por la menor cantidad de código que se tenga que escribir y la menor cantidad de modificaciones que se hara al código existente; además de la facilidades que trae consigo el desarrollo sobre una Arquitectura Abierta. Sólo se trabajara en tres de los cinco subsistemas que conforman un Sistema de Control Distribuido que son:

   - Estacion de Operaciones
   - Controladores
   - Red de Comunicaciones

4. Lograr la mayor cantidad de código reutilizable, tanto para lograr la extensibilidad de la aplicación como para reutilizar código en otras aplicaciones similares. Para ello se utilizara técnicas de programación orientadas a objetos; además de la construcción de una Interface de Usuario totalmente gráfica, lo que resultara en una identificación más transparente de los eventos a controlar.

5. Permitir que la aplicación que se construya sea flexible, de bajo costo y de arquitectura abierta y extendible tanto vertical como horizontalmente [1].

Funciones Principales

1. Permitir la ejecución de la simulación de un proceso representado por su modelo matemático y la de su Controlador; además de poder visualizar los resultados del proceso.

2. Permitir la ejecucion de la aplicación en una red de computadoras usando el protocolo TCP/IP y la filosofía de Cliente/Servidor.

3. Lograr la visualización de la aplicación en múltiples estaciones de trabajo mientras se ejecuta en sólo una de ellas a través de la multiplexación de la pantalla de la aplicación a varios terminales conectados en red.

[1] Se refiere a la forma como se extiende la red de información fisicamente

Aspectos de Funcionamiento

La aplicación a desarrollar, "lee"las señales del proceso elegido usando la interface de comunicación del UNIX representada por los sockets [2] bajo el dominio del protocolo TCP/IP. Por medio de una interface de usuario gráfica se eligen tanto el proceso a controlar como la configuración de parametros del proceso, y la elección del anfitrión donde se ejecutará la aplicación. Para la aplicación, es transparente la planta a controlar, en el sentido de que la lógica de la aplicación no cambia por el proceso elegido.

Asimismo mediante una interface de usuario se configura las señales del proceso que se quieren visualizar para analizar la información que de ellas se pueda extraer. Para ello se necesita una pantalla de visualización flexible para mostrar gráficos en 2D.

La simulación del proceso es la parte que genera la base de tiempo para el proceso, por ello el subsistema de simulación servira para sincronizar las operaciones de la aplicación. El subsistema de simulación recibe la señal de control generada por el subsistema de control que junto con las condiciones iniciales del proceso se utilizan para resolver las ecuaciones diferenciales ordinarias que gobiernan la dinámica de la planta. Por intermedio de la IU del subsistema de simulación se eligen las condiciones iniciales del proceso y parámetros de la planta.

El subsistema de control del proceso es el encargado de generar la señal de control que alimenta la planta. Esta función la realiza usando las variables de estado del modelo matemático de la planta o con la señal de salida; si sólo tiene acceso a las salidas de la planta. La IU del subsistema de control permite seleccionar el tipo de Controlador a utilizar y sus parámetros correspondientes.

4.2.2 Estimaciones del Proyecto

Para nuestro caso se utilizan sólo dos de las opciones descritas anteriormente, las técnicas de descomposición y el desarrollo de un modelo empírico.

Técnicas de Estimación

La primera estimación hecha para calibrar cada elemento del software fue la estimación de las lineas de código para obtener posteriormente al final del proyecto métricas de productividad. El nivel de detalle a partir del cual se hizo el cálculo fueron los subsistemas componentes del software SCD_Avanzado. El cálculo se realizó a partir de la siguiente relación [18]: (sigue una probabilidad beta)

$$E = (a + 4m + b)/6$$

[2] punto de comunicación por el cual un proceso puede emitir o recibir información

donde :

E	media ponderada de las estimaciones
a	optimista
b	mas probable
m	pesimista

La segunda estimación realizada fue la del esfuerzo requerido para cada tarea de software (por ejemplo : personal/mes, coste/unidad-esfuerzo). Aquí el nivel de detalle utilizado fueron los subsistemas del software. Hechas las dos estimaciones en forma independiente tenemos dos resultados que se pueden comparar y unir.

Para propósitos de tener más argumentos de confiabilidad se realizó una estimación empírica del proyecto utilizando el modelo de estimación COCOMO [19] [3]. Las características del modelo COCOMO que se utilizarón fueron considerando el proyecto de software del tipo modo semi-acoplado (proyectos de software Intermedios en tamaño y complejidad) en los equipos, con variados niveles de experiencia, satisfaciendo niveles medios de rigidez. La ecuación del modelo COCOMO utilizada es:

$$E = a_i(LDC)exp(b_i) * FAE$$

donde :
FAE : factor de ajuste del esfuerzo.
$b_i = 1.12$; $a_i = 3.0$ (constantes para un modelo semiacoplado).
El FAE se calcula a partir de la evaluación de un conjunto de atributos conductores de coste, que pueden agruparse en cuatro categorias principales: atributos del producto, atributos del hardware, atributos del personal y atributos del proyecto.

Estimaciones

La estimación realizada de las lineas de código se muestra en la tabla 4.2.2. La estimación del esfuerzo requerido se muestra en la tabla 4.2.2.

Aplicando el uso del modelo COCOMO al proyecto de software SCD_Avanzado tenemos:

$$E = 3.0(16485)exp(1.12)$$

El esfuerzo requerido:

$$E = 158 \ personas - mes$$

Y la duración del Proyecto:

$$D = 2.5(158)exp(0.35)$$

[3] Constructive Cost Model (Modelo Constructivo de Costos)

	Optimista	Más posible	Pesimista	Esperada
IUP	1000	1500	2300	1600
Controlador	800	1000	2500	1050
Simulador	1000	1300	3000	1267
Visualizador	8000	12000	15000	11834
Comunicaciones	500	700	1000	734
Total				16485

Tabla 4.1: Estimación de LDC

	Anal. Requi.	Diseño	Codificación	Prueba	Total
IUP	2.0	1.5	1.0	0.5	5.0
Controlador	1.0	1.0	1.0	0.5	3.5
Simulador	1.0	1.5	2.0	1.5	6.0
Visualizador	1.0	1.5	1.0	1.0	4.5
Comunicaciones	2.0	1.0	1.5	1.0	5.5
Total	7.0	6.5	6.5	4.5	24.0

Tabla 4.2: Estimación de Esfuerzo

$$D = 14.7 \ meses$$

El número de personas participantes:

$$N = E/D = 10 \ personas$$

Para nuestro caso particular se decidió emplear sólo cinco personas y ampliar por tanto la duración del proyecto. Los resultados obtenidos anteriormente son para tener una idea de los costos de trabajo en tiempo y personal que involucra el desarrollo del software, sin que ello signifique que sean resultados que dirijan el proyecto.

4.2.3 Riesgos del Proyecto

El análisis del riesgo se realiza ante la necesidad de preveer las acciones a tomar en caso que el proyecto de software fracase ó este a punto de hacerlo. Visto asi lo que hay que identificar son los riesgos que pueden hacer que fracase el proyecto, como afectarán al exito global y a los plazos los cambios en los requisitos; en las herramientas utilizadas, en las computadoras destino de la aplicación.

Análisis de Riesgos

- **Identificación**

  La tarea aquí es enumerar los riesgos concretos del proyecto. Es decir los problemas potenciales de agenda, presupuesto, personal y de requisitos y su repercusión en el proyecto de software.
  A pesar de que algunos riesgos son imposibles de predecir de antemano es importante identificar todos los riesgos obvios en el proyecto de software. De esta manera identificamos los siguientes riesgos potenciales:

  1. El personal del que se dispone para el proyecto no tiene experiencia en programación orientada a objetos, ni en diseño de software sobre plataforma Unix.
  2. El personal encargado del proyecto es en su totalidad estudiantes de la Universidad, y su participación está comprometida en base al tiempo que ofrecen fuera de hora de estudios; además de existir la posibilidad de comenzar a realizar practicas pre-profesionales de la especialidad en una empresa.
  3. La probabilidad de que el personal participando del proyecto decida abandonar por motivos de trabajo el proyecto es alta debido a que cursan ciclos de estudio avanzados, proximos a concluir la carrera.
  4. Un problema a resolver es como realizar la trasferencia de datos a tráves de maquinas con arquitecturas diferentes.
  5. Una dificultad en la etapa de diseño que podría existir esta relacionada con la construcción de las interfaces de usuario en lo referente a la elección del lenguaje de programación a utilizar.

- **Estimación del riesgo**

  Aquí el propósito es evaluar cada riesgo de dos formas: la probabilidad de que el riesgo sea real y las consecuencias de los problemas asociados con el riesgo si aparece.

  1. La probabilidad del riesgo que supone contar con participantes con conocimiento de POO nulo y falta de experiencia en diseño de software es alta, ello involucra que los tiempos para la ejecución del proyecto pueden sufrir ampliaciones que provocaria atrasos y extensión del tiempo de ejecución del proyecto. La falta de conocimiento de POO hace preveer que la etapa de análisis tendrá que ser muy sólida para evitar retrocesos en el desarrollo del software.

2. El riesgo de que los participantes dediquen poco tiempo al proyecto y en el peor de los casos abandone el proyecto tiene una probabilidad alta, debido a que el tiempo que dedican los estudiantes al proyecto no es constante, ademas que es probable su alejamiento del proyecto.
   Ello provocaria que módulos empezados por una persona tengan que ser culminados por otras personas lo que causaria a la larga problemas de diseño cada vez más dificiles de resolver.
3. El problema relacionado con la transferencia de datos a tráves de maquinas con arquitecturas diferentes tiene una probabilidad moderada ya que existe software especializado para ello.
   Lo importante es determinar si la interface de comunicación será una libreria de software disponible en internet o tendrá que ser construida con los costos en tiempo que ello involucraria.
4. La elección del lenguaje de programación para la construcción de la interface de usuario tiene una probabilidad moderada debido a que existen varias posibilidades para decidir y lo que más interesa es saber con cual se tomara menos tiempo construir las interfaces.

- **Gestión del Riesgo**

  Aquí el objetivo es tratar de dar una serie de recomendaciones para los riesgos calculados en los puntos anteriores. Para prevenir el caso de abandono del proyecto de uno o más integrantes se hace necesario que los tiempos de ejecución del proyecto sean cortos y que se realizen reuniones de evaluación donde todos los participantes tengan participación directa de tal manera que si un integrante se retira del proyecto, otra persona este en capacidad de tomar su trabajo en un tiempo corto, afectando el desarrollo del proyecto de forma minima.
  Se hace necesario considerar en los tiempos del proyecto un periodo de aprendizaje para programación orientada a objetos, utilizando el compilador de la plataforma Unix.
  Asimismo se recomienda considerar un periodo de prueba y aprendizaje de los diferentes lenguajes de programación para la construcción de interfaces gráficas considerandose para tal fin el lenguaje Xview y la libreria InterViews, proponiendose la construcción de prototipos similares y comparar las ventajas y desventajas de cada una.
  Para el problema de transferencia de datos entre arquitecturas diferentes se hace necesario un periodo de aprendizaje de la libreria DTM y determinar si sirve para los propositos del proyecto, de lo contrario considerar un periodo adicional para

la construcción de un programa que resuelva el problema, utilizandose para ello las facilidades de comunicaciones que ofrece el S.O. Unix.

4.2.4 Agenda

Estructura de descomposición de trabajos del Proyecto

Los trabajos a desarrollar del proyecto se realizarón en base a los siguientes:

1. Solución de los modelos matemáticos de los procesos elegidos. Aquí las tareas se dividieron en:
   - Etapa de Aprendizaje.
   - Etapa de Planteamiento del problema.
   - Etapa de Solución del problema.
   - Etapa de Simulación.
   - Etapa de Documentación de trabajo.
2. Sistema de Software
   - Estudio de trabajos anteriores y aprendizaje de herramientas de desarrollo de software.
   - Lograr el Plan de Proyecto de software.
   - Realizar un documento de especificaciones del software.
   - Lograr una metodología de diseño para construir la aplicación.
   - Construir en código fuente cada uno de los subsistemas a realizar.
   - Depuración de la aplicación.
   - Prueba de Integración.
   - Documentación del trabajo realizado y escribir documentación para el usuario de la aplicación.

4.2.5 Recursos del Proyecto

Personal

El personal destinado a trabajar en el Proyecto esta conformado inicialmente por seis personas cuyos perfiles profesionales se dirigian como en la tabla 4.2.5 y la tabla 4.2.5.

Participante	Conocimiento Lenguaje C++	Conocimiento Diseño OO
LHB	Si	Si
RDC	No	No
OTC	No	No
JVG	No	No
EAL	Si	No

Tabla 4.3: Perfil de Participantes de Proyecto

	Experiencia	Niv. Academico	Conoc. Prog.	Disponibilidad	Exper.
LHB	Excelente	Ph.D.	Excelente	15 hor/sem	Si
RDC	Regular	Estudiante	Regular	15 hor/sem	No
OTC	Regular	Estudiante	Regular	15 hor/sem	No
JVG	Regular	Estudiante	Bueno	15 hor/sem	No
EAL	Regular	estudiante	Regular	15 hor/sem	No

Tabla 4.4: Perfil de Participantes de Proyecto

Hardware y Software

El hardware con el que se dispone está compuesto por los recursos del laboratorio:

1. Una red Ethernet TCP/IP formada por tres anfitriones sobre una plataforma del S.O. Unix.
   - 1 SunSparcClassic v5 con S.O. SVR4
   - 1 PC 486 compatible con S.O. Linux (distribución Red Hat)
   - 1 HP Vectra RS/20 con S.O. Linux
2. Con respecto al software tenemos:
   - Versión estudiantil de MATLAB para ambiente Windows.
   - Compiladores GNU C++ y C++ de Sun.
   - Librerias gráficas Xview, Xlib instaladas en toda la red.
   - Bibliografía del fabricante y del distribuidor, además de libros sobre el lenguaje de programación, el S.O. Unix y el lenguaje Xview y Xlib.
   - Aplicación Gnuplot herramienta disponible en internet que sirve para el ploteo de señales en 2D y 3D.

- Aplicación Octave, herramienta de simulación matemática disponible en internet que representa un clone de MATLAB para plataforma Unix.
- Aplicación Xmx, disponible libremente en internet y cuyo próposito es multiplexar la salida gráfica de una aplicación a tráves de un sólo proceso servidor de pantalla.
- Aplicación DTM, cuyo propósito es disponer de una interface trasparente para la trasferencia de datos entre arquitecturas diferentes de computadoras.
- Libreria gráfica InterViews para la construcción de interfaces de usuario.

4.3 Análisis de Requisitos del Sistema de Software

4.3.1 Introducción

Ambito y Propósito

Las especificaciones del Sistema de Control Distribuido (SCD_Avanzado) tienen como propósito principal describir la función y rendimiento del software, asi como las restricciones que gobernarán su desarrollo. Aquí se describe la información que sirve de entrada y salida del software.

4.3.2 Descripción Funcional y de Datos

1. **Arquitectura del Sistema**

   El software SCD_Avanzado consta de cuatro Subsistemas : (1) Interface de Usuario, (2) Visualizador, (3) Simulador y (4) Controlador. Los subsistemas son autónomos y se sincronizan mediante mensajes a tráves del subsistema de comunicaciones implementado bajo el dominio del protocolo TCP/IP e independiente de los cuatro Subsistemas anteriores. El diagrama de bloques de la figura 4.1 muestra los subsistemas y flujo de datos del software. Para facilitar las labores del modelización de la arquitectura del sistema, es decir, crear un modelo que represente las interrelaciones entre los distintos elementos del sistema se utilizan dos tipos de plantilla de arquitectura:

   - Diagrama de Contexto de la Arquitectura (DCA)
     El DCA establece los limites de información entre los que se está implementando el sistema y el entorno en el que va a funcionar el sistema. El Diagrama de Contexto de la Arquitectura (DCA) [21] define todos los productores externos de información usados por el sistema, todos los consumidores externos

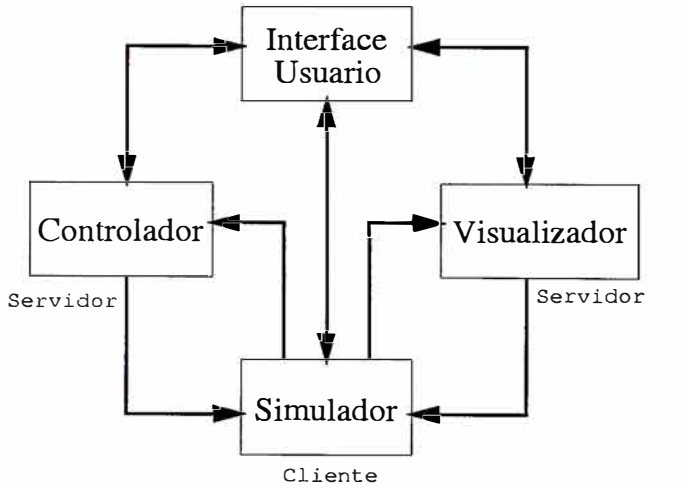

Figura 4.1: Arquitectura y Flujo de Datos del Sistema de Software

de información creada por el sistema, y todas las entidades que se comunican por medio de la Interface de Usuario. Los elementos de cada subsistema son ubicados en un gráfico con cinco regiones de procesamiento. Estas regiones son: (1) Interface Usuario, (2) Entrada, (3) Funciones de Procesamiento y Control, (4) Salida y (5) Diagnostico. Este modelo permite crear una jerarquía donde el DCA esta en el nivel más alto.

- Diagrama de Flujo de la Arquitectura (DFA)
  El Diagrama de Flujo de la Arquitectura muestra los subsistemas principales y las lineas importantes de flujo de información (control y datos), descomponiendo el sistema en módulos con su respectivo flujo de información. Los módulos también estan ubicados en una plantilla igual a la del DCA con las cinco regiones de procesamiento.

Subsistema Interface Usuario (IU)

El subsistema Interface Usuario (IU) permite al usuario interactuar con el software y acceder a los otros tres subsistemas seleccionando los anfitriones donde estos se ejecutan. El DCA de la IU se muestra en la figura 4.2 y el DFA en la figura 4.3. Los subsistemas envian información a la IU para que la presente en la consola del usuario. Cuando el usuario termina de ingresar los datos, estos son enviados a los subsistemas respectivos. La IU se divide en tres módulos : (1) Manejador de Interfaces, (2) Comunicaciones y (3) Diagnóstico.

El módulo Manejador de Interfaces es el encargado de recibir información de los otros subsistemas para mostrarla en la consola del usuario y de enviar la información de configuración y de parámetros. El módulo de Comunicaciones es el encargado de

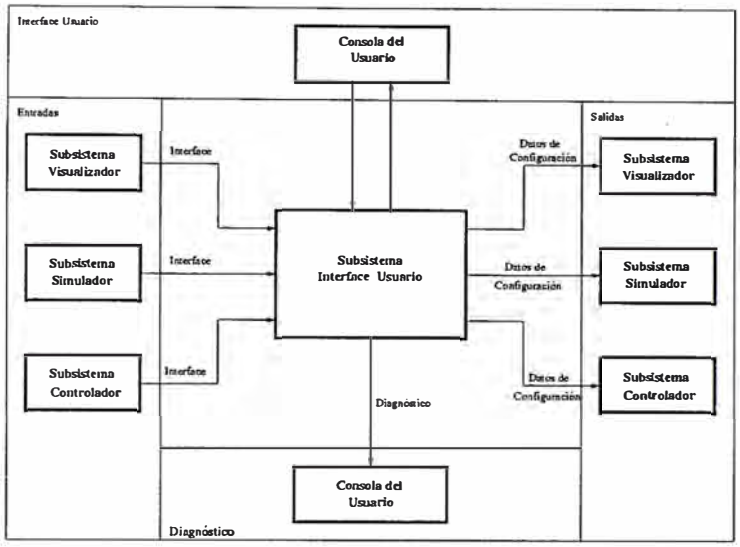

Figura 4.2: DCA del Subsistema Interface Usuario

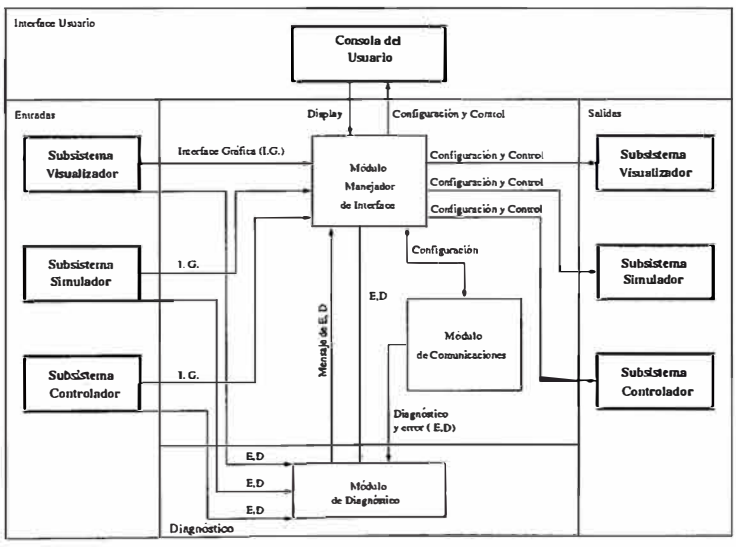

Figura 4.3: DFA del Subsistema Interface Usuario

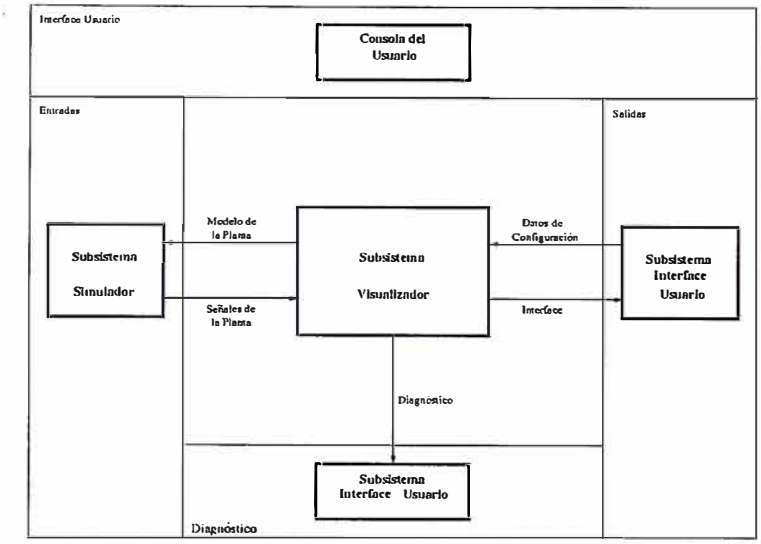

Figura 4.4: DCA del Subsistema Visualizador

establecer el canal de comunicación con las otras computadoras de la red donde se van a ejecutar los otros subsistemas. El módulo de Diagnóstico comunica al usuario los mensajes de error que se generan durante la operación del sistema.

Subsistema Visualizador (VI)

Se encarga de presentar información gráfica del proceso que se esta controlando. Esta información consiste en gráficos de las señales del proceso. El DCA de este subsistema se muestra en la figura 4.4 y el DFA en la figura 4.5. El usuario define que señales del proceso graficar (variables de estado, control y salida). El subsistema Visualizador se ha dividido en dos módulos (1) Graficador y (2) Comunicaciones. El Graficador dibuja las señales que recibe del Simulador. El módulo de Comunicaciones, al igual que el caso anterior, es el que establece un canal de comunicaciones con los otros subsistemas.

Subsistema Simulador (SI)

El subsistema Simulador se encarga de simular la planta elegida. El DCA del SI se muestra en la figura 4.6 y el DFA en la figura 4.7. La Interface de Usuario Principal permite al usuario u operador administrar el subsistema Simulador y dar las señales de inicio y parada del subsistema. El Controlador envia la señal de control (U) al subsistema Simulador, que la usa para resolver las ecuaciones diferenciales que modelan la planta seleccionada y generar las variables de estado (X) y de salida (Y) que serán devueltas al Controlador para obtener una nueva señal de control (U). El Simulador envia las variables de estado (X), salida (Y) y la señal de Control (U) al Visualizador para su posterior visualización. Los módulos del subsistema Simulador son (1) Interface

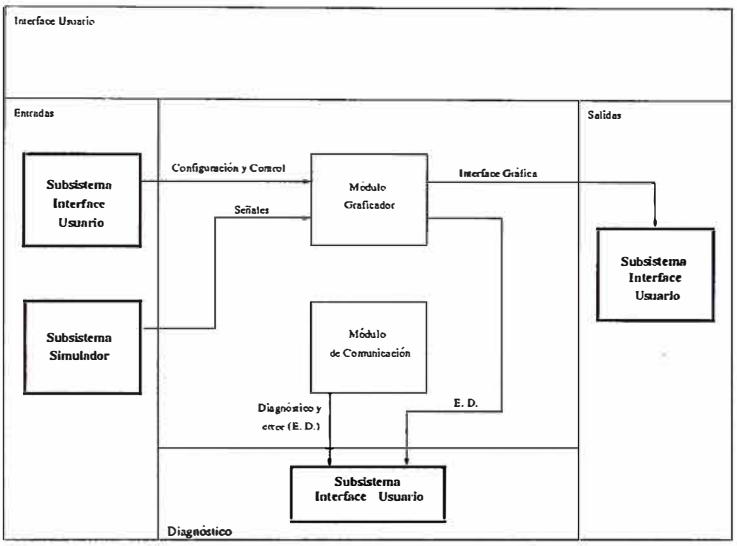

Figura 4.5: DFA del Subsistema Visualizador

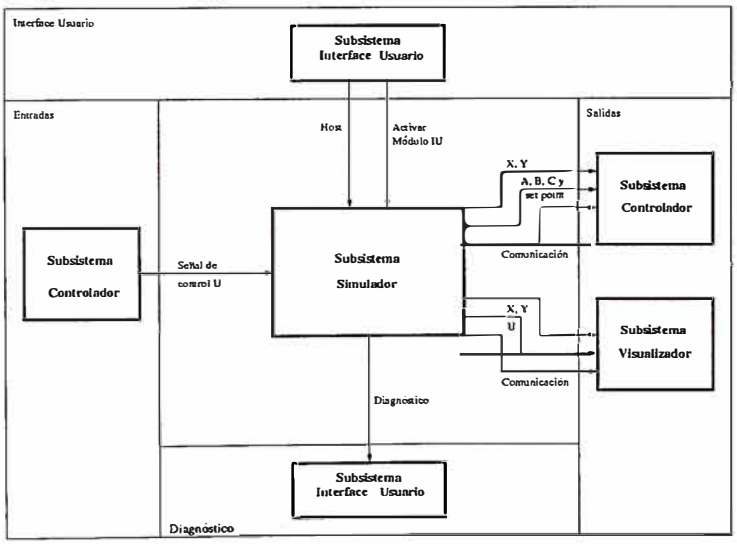

Figura 4.6: DCA del Subsistema Simulador

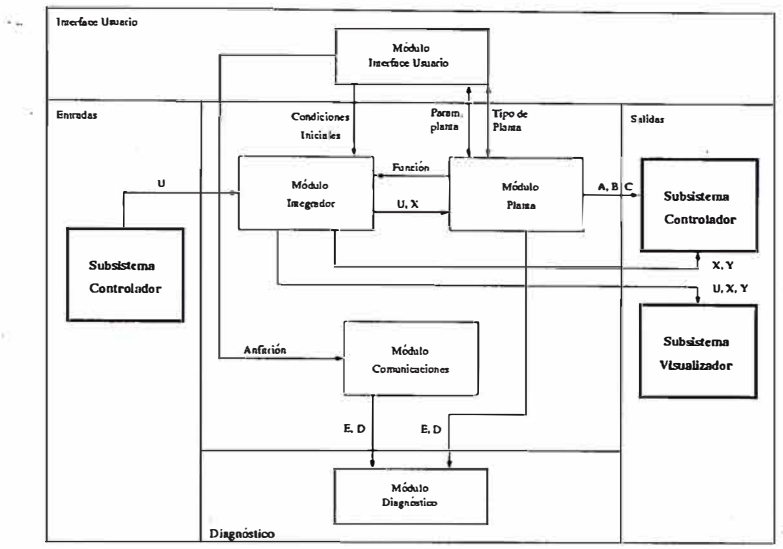

Figura 4.7: DFA del Subsistema Simulador

de Usuario, (2) Planta, (3) Integrador y (5) Comunicaciones.

El módulo Interface de Usuario permite tomar mando sobre algunos aspectos de este subsistema tales como ingresar los parámetros correspondientes de la planta a controlar, condiciones iniciales del proceso, tiempo total de simulación y variables de salida. El módulo Planta en base a los parámetros que son ingresados desde la IU, genera el modelo matemático de la Planta seleccionada. El módulo Integrador resuelve las ecuaciones diferenciales del modelo matemático obtenido del módulo Planta, teniendo como entradas la señal de control (enviada por el Controlador) y las variables de estado. Este módulo obtiene las variables de estado actualizadas y las envia al Controlador, asimismo permite obtener las variables de salida y las envia junto con las variables de estado y control al Visualizador. El módulo Comunicaciones hace posible la correcta transmisión y es el encargado de sincronizar el trabajo del Simulador, Controlador y Visualizador de tal modo que los mensajes transmitidos entre los subsistemas guarden una secuencia prestablecida.

Subsistema Controlador (CO)

El subsistema Controlador tiene como función generar la señal de control U para conseguir el funcionamiento deseado de la planta a controlar. El DCA del subsistema CO se muestra en la figura 4.8 y el DFA en la figura 4.9. Este recibe las variables de estado y salida de la planta a través del subsistema SI. El subsistema CO está formado de tres módulos : (1) IU del Controlador, (2) Controlador, (3) Diagnóstico.

El módulo Interface de Usuario permite elegir el tipo de Controlador, dar valores a sus parametros y controlar la operación del Subsistema. El módulo Controlador recibe

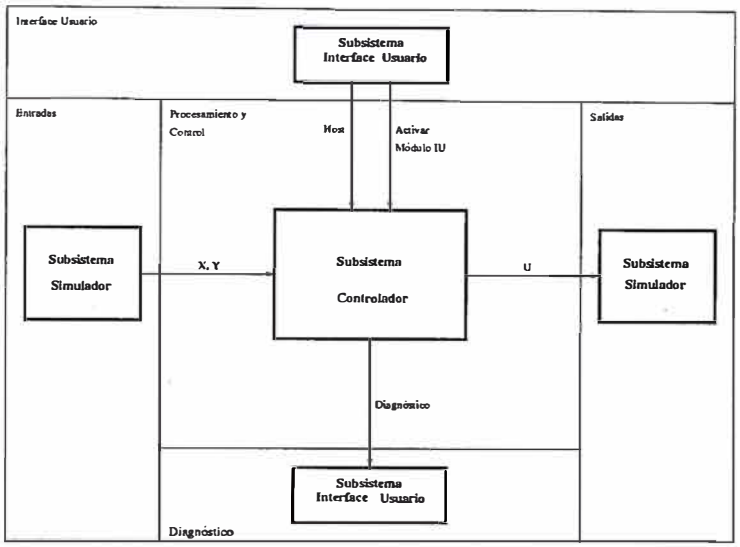

Figura 4.8: DCA del Subsistema Controlador

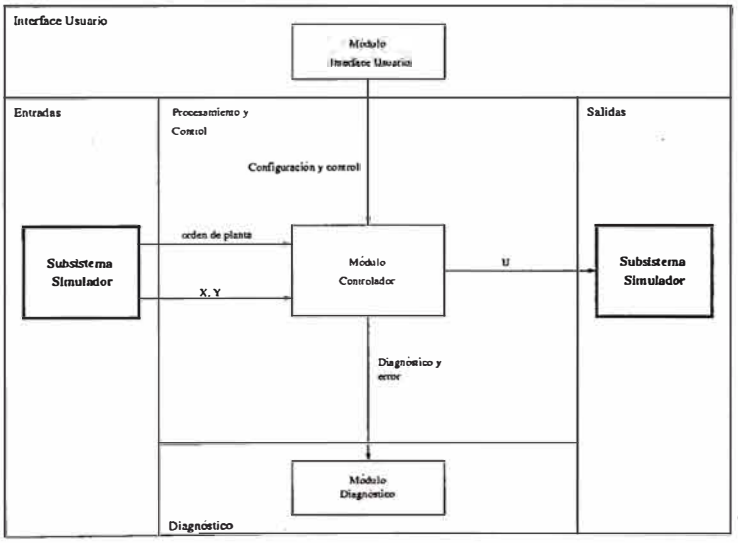

Figura 4.9: DFA del Subsistema Controlador

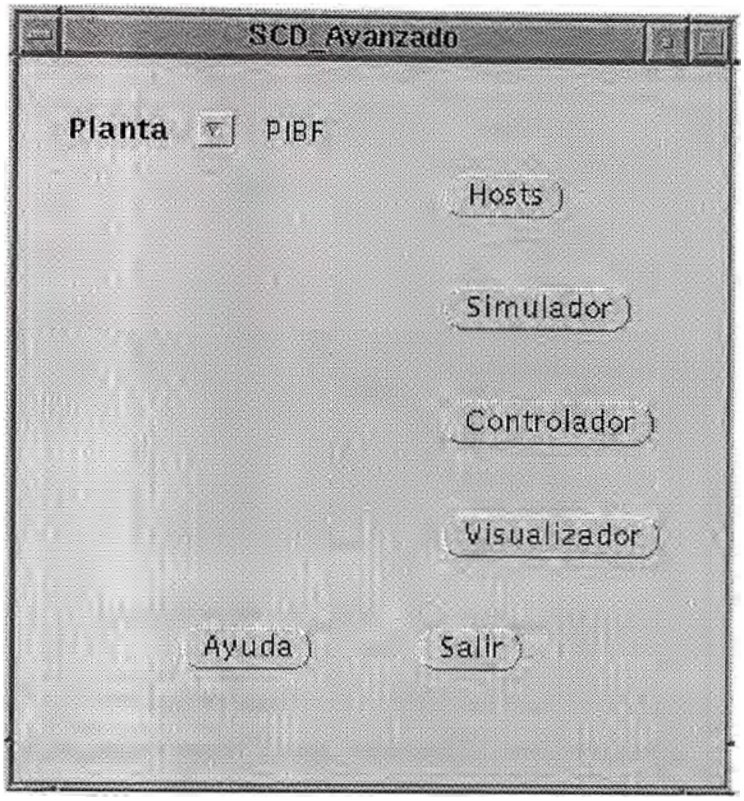

Figura 4.10: Ventana principal del SCD_Avanzado

las variables de estado (o salida) de la planta del subsistema Simulador. La señal de control generada a partir de las variables de estado por el Controlador es enviada al subsistema Simulador. El módulo Diagnóstico envia los mensajes generados por el Controlador a la Interface de Usuario para su visualización.

4.3.3 Descripción de los Subsistemas

Especificación del Diagrama de Arquitectura para el Subsistema

A continuación se describe cada uno de los módulos de los subsistemas, detallandose la Interface de Usuario Principal de cada subsistema, que se realizará utilizando el lenguaje de programación gráfico Xview, mencionando además las características de funcionamiento de los módulos y objetos que los conforman.

- **Subsistema Interface Principal de Usuario (IPU):**

  Es la Interface Principal con el usuario que le permite controlar y configurar la aplicación. La ventana principal se muestra en la figura 4.10 y tiene las siguientes funciones:

Figura 4.11: Ventana de selección de anfitrión del SCD_Avanzado

1. Seleccionar los anfitriones (estaciones de trabajo) donde se van a ejecutar los subsistemas: Controlador, Simulador y Visualizador. La ventana donde se selecciona el anfitrión se muestra en la figura 4.11.
2. Seleccionar el tipo de planta donde se desea aplicar el control.
3. Invocar los otros tres subsistemas presionando los botones Controlador, Simulador, y Visualizador.
4. Descripción de instrucciones de usuario que se obtienen al presionar el botón Ayuda.
5. Mensajes para prevenir al usuario de posibles errores en el manejo de la aplicación.

- **Subsistema Visualizador VI:**

Este subsistema permite visualizar las señales involucradas en el proceso en una o más ventanas gráficas, permitiendole al usuario elegir las opciones más adecuadas para el gráfico. La interface de usuario del Visualizador **??** permite ejecutar las siguientes funciones:

1. Definir varias ventanas gráficas que contengan una o más señales. Estas ventanas están en una lista que incluye el titulo del gráfico. A esta lista se le puede añadir, eliminar o editar elementos.
2. Cada ventana gráfica tiene un conjunto de parámetros de configuración (colores de señales, limites del gráfico, etiquetas de texto, el enrejado del gráfico, que se definen en una ventana adicional).

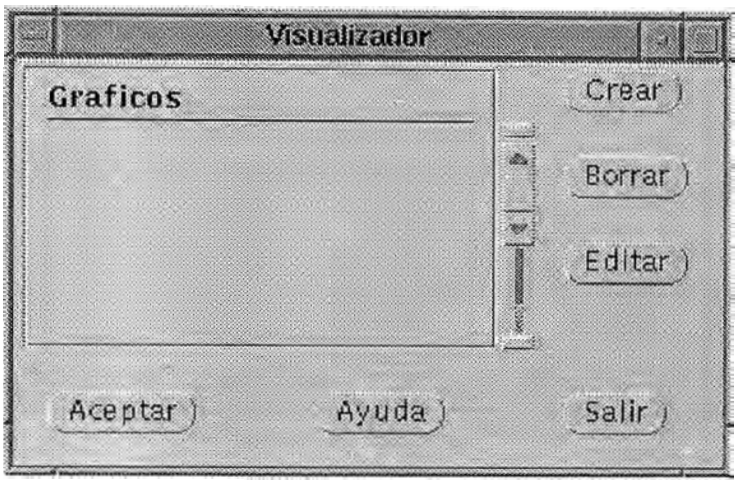

Figura 4.12: Ventana principal del Subsistema Visualizador

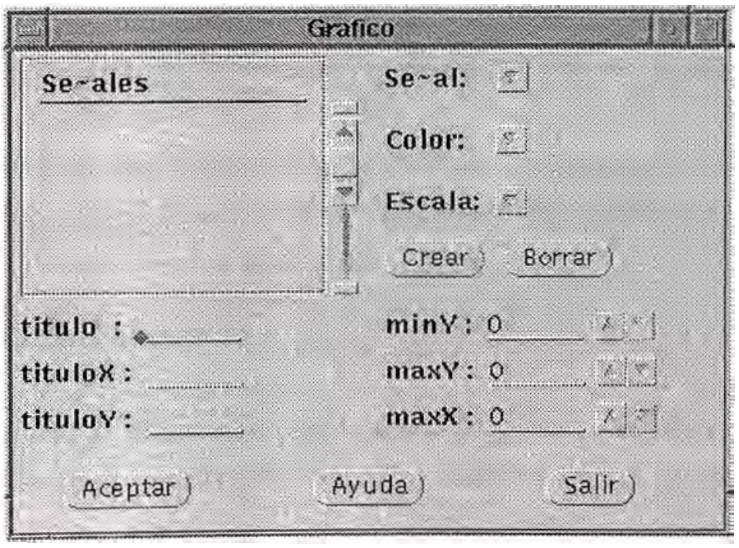

Figura 4.13: Ventana de configuración de gráficos de señales

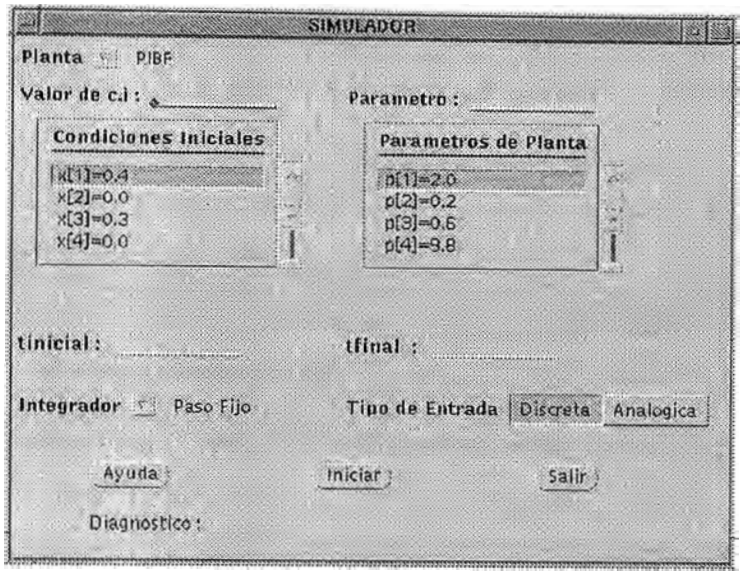

Figura 4.14: Ventana principal del Subsistema Simulador

3. Cada vez que se crea o modifica una ventana gráfica, esta se selecciona en la ventana principal, invocandose una segunda ventana que se muestra en la figura 4.13
4. A esta lista se puede añadir o eliminar señales. Para cada señal se le asigna un color y un nombre proveniente de las listas que mantiene el sistema por el tipo de planta y controlador que se están simulando.

El módulo Graficador permite almacenar información necesaria que define una señal a ser graficada en la ventana de dibujo y tiene las rutinas gráficas que se requieren para ello.
Esta información básicamente son los titulos y etiquetas del gráfico, el tipo de linea, el color y los puntos a graficar, el enrejado del gráfico, los limites del marco de dibujo y dimensiones de la ventana gráfica.

- **Subsistema Simulador SI:**

Permite realizar la simulación de la planta elegida. Contiene un módulo de Comunicaciones para intercambio de información con el Controlador e Interface Principal.
El módulo Interface de Usuario del Simulador se muestra en la figura 4.14. Este módulo configura el Simulador, pudiendo seleccionar las siguientes opciones:

1. Parámetros del proceso a controlar.
2. Condiciones iniciales del proceso.

3. Tiempo de simulación.
4. Tipo de entrada a la planta a controlar (analógica o discreta).
5. Tipo de integración.
6. Botones para iniciar simulación, para solicitar Ayuda y para salir de la interface.

El módulo Planta es el que contiene la información que define el tipo de proceso. Contiene la siguiente información:

Parámetros fijos: Vienen a ser el nombre de la planta, orden del sistema, el número de entradas.

Parámetros de entrada: Son parámetros definidos en el módulo Planta, e ingresados desde la Interface de Usuario.

Parámetros de salida y el modelo matemático de la planta.

Las funciones principales del módulo Planta son:

1. Informar los tipos de planta disponibles al módulo Interface de Usuario del Simulador.
2. Enviar los parámetros que requiere la planta escogida hacia el módulo Interface de Usuario.
3. Recibir las variables de control del módulo Integrador, para evaluar la función del modelo matemático.
4. Enviar la función evaluada del paso anterior hacia el módulo Integrador.

El módulo Integrador es el encargado de resolver las ecuaciones diferenciales ordinarias que modelan la planta seleccionada usando el método Runge-Kutta de cuarto orden. Los resultados son enviados al Controlador y al Visualizador. Recibe la señal de control y las variables de estado iniciales del Controlador y las actualiza para cada tiempo de integración almacenando los resultados para su envio al Visualizador.

- **Subsistema Controlador CO**

Implementa el algoritmo de control usando la información de estado recibida del Simulador. El módulo Interface de Usuario del Controlador permite controlar la selección del tipo de Controlador a usar y sus parámetros. Esta Interface de Usuario se muestra en la figura 4.15. Las funciones generales de este módulo son:

Figura 4.15: Ventana principal del Subsistema Controlador

1. Control del inicio y parada de la operación del módulo Controlador.
2. Recibir el orden de la planta, con la finalidad de que los parámetros del Controlador sean solicitados al usuario en forma consistente.
3. Listado y selección del tipo de Controlador y elección de sus parámetros dependiendo del Controlador seleccionado y del orden de la planta.
4. Contiene un módulo de Comunicaciones que le permite comunicarse con los subsistemas Interface Principal y Simulador.

El módulo Controlador es el que genera la señal de control utilizando diferentes métodos de acuerdo al Controlador seleccionado. Sus características son:

1. Contiene los tipos de Controladores del sistema y sus respectivos parámetros de acuerdo a la planta elegida.
2. Recibe las señales de control de inicio/parada del módulo Interface de Usuario del Controlador.
3. Recibe las variables de estado de la planta desde el subsistema Simulador.
4. Genera la señal de control, mediante operaciones efectuadas sobre las variables de estado o salida de la planta, que dependen del tipo de controlador seleccionado.
5. Envia la señal de control al subsistema Simulador.
6. Contiene un módulo que le permite establecer un canal de comunicación con el subsistema Simulador.

- **Subsistema Comunicaciones :**

Este subsistema no posee una interface de usuario directa con el usuario pero

interviene en todo el intercambio de información que se produzca entre los diferentes subsistemas. Hace posible la comunicación entre dos o más procesos que pertenecen a una o más computadoras. La filosofía del subsistema es Cliente/Servidor usando el protocolo de comunicación TCP/IP (protocolo orientado a conexión). Para el diseño e implementación del subsistema se recomienda el uso de sockets, mecanismo de comunicación proporcionado por Unix.
Las principales funciones de este subsistema son:

1. Identificar el anfitrión al que se reconoce como Servidor y el anfitrión al que se reconoce como Cliente.
2. Permitir un tiempo de espera prudencial tanto del Servidor como del Cliente, el uno por el otro hasta que la conexión se establezca; de lo contrario abandonar la aplicación.
3. Realizar la transferencia de datos de punto flotante entre computadoras de arquitecturas diferentes realizando para ello una codificación y decodificación de los datos. Codificando al enviar datos y decodificando al recibir datos.

4.4 Diseño e Implementación del Sistema de Software

Los cuatro subsistemas: Interface Usuario, Simulador, Visualizador y Controlador han sido implementados en lenguage C^{++} [22], utilizando técnicas de programación orientadas a objetos [23]. Cada subsistema consta de un conjunto de módulos descritos en la sección anterior. Estos módulos estan formados por un conjunto de objetos cuya estructura jerarquica de clase se muestra en la Figura 4.16. Esta figura sólo muestra los principales objetos que conforman el SCD_Avanzado.

4.4.1 Lenguaje de Programación C^{++}

C^{++} fue originalmente un precompilador, similar al preprocesador de C. El precódigo, es leido por el precompilador C^{++}, encontrandose en un archivo con extensión .cc o .cpp . Visto asi C^{++} es un superconjunto de C que ofrece todas las posibilidades de C y más. Esto permite una transición fácil de C a C^{++}. Las ventajas que encontramos de usar el lenguaje C^{++} son:

- Nuevos programas pueden ser desarrollados en menos tiempo, pues el código antiguo puede ser reusado.
- Crear y usar nuevos tipos de datos es más fácil que en C.
- El manejo de memoria bajo C^{++} es más fácil y transparente.

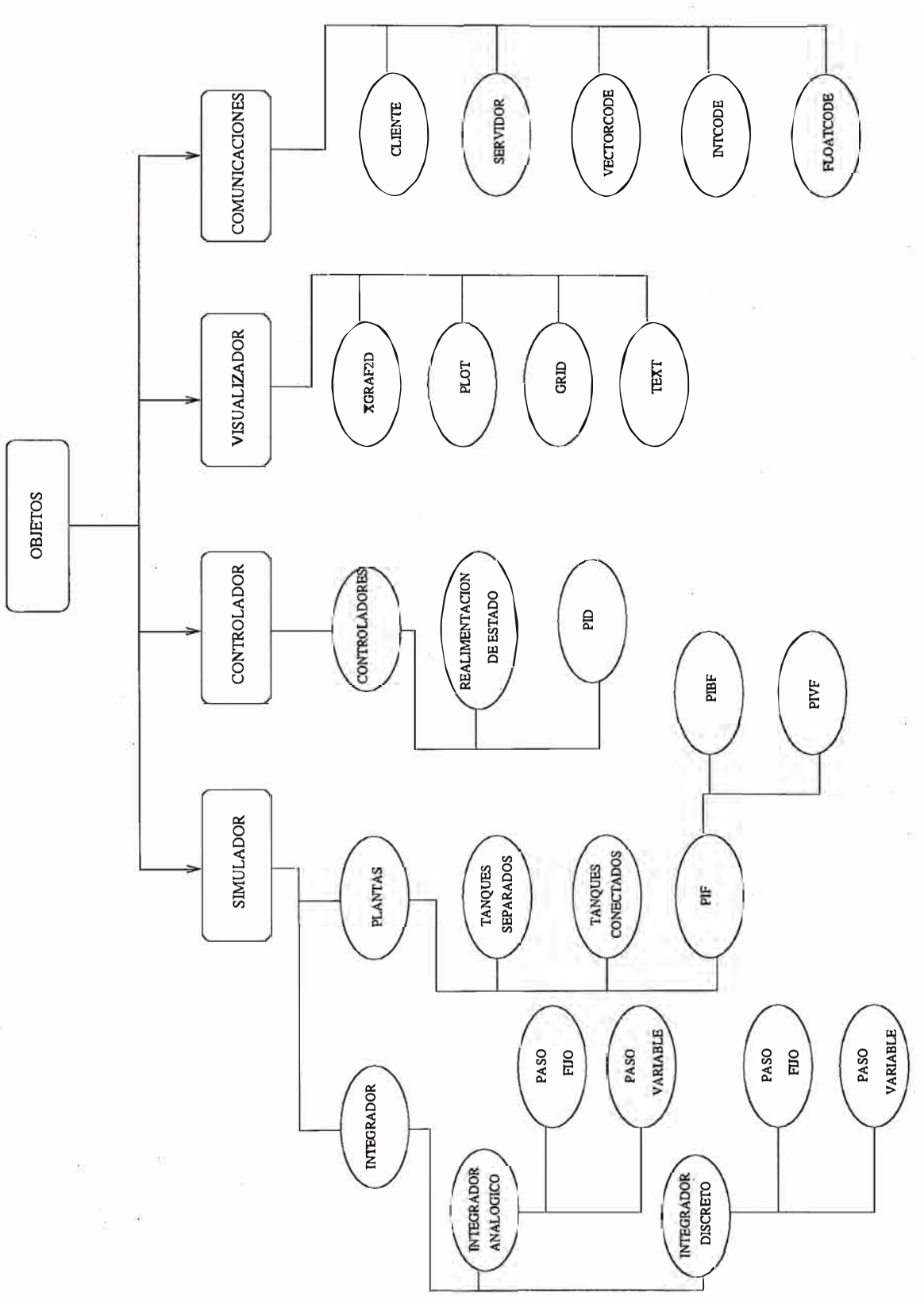

Figura 4.16: Estructura Jerarquica de los principales objetos del SCD_Avanzado.

- El ocultamiento de información es más fácil de implementar C++

Sin embargo todas estas pretensiones del C++ son también faciles de obtener en lenguaje C, por lo que concluimos que C++ en particular y POO en general no son soluciones en todos los problemas de programación. Sin embargo C++ ofrece algunas posibilidades de sintaxis elegante.

4.4.2 Programación Orientada a Objetos

La programación orientada a objetos permite usar una estrategia diferente de enfrentar problemas de programación que la brindada en C. La estrategia en C es llamada diseño estructurado, donde cada problema es dividido en subproblemas y este proceso es repetido hasta que las subtareas puedan ser codificadas.

Por el contrario el diseño orientado a objetos comienza por identificar las claves del problema. Estas luego son pasadas a un diagrama de jerarquia interno. Estas claves pueden ser los objetos en la implementación y la jerarquia define la relación entre estos objetos. El término objeto es utilizado para definir una estructura de datos definida.

4.4.3 Implementación Subsistema Interface Usuario

Este subsistema consta de dos partes claramente definidas, la primera referida a la implementación de la Interface Principal de Usuario en lenguaje XView [25] y que permite al usuario identificar en un sistema de ventanas, los subsistemas de los que dispone y la planta que puede elegir. Representa la entrada al sistema de software y la salida final cuando se abandona la aplicación.

La segunda es la parte de comunicaciones que permite configurar por pantalla los anfitriones donde se ejecutará cada uno de los subsistemas, planificandose además la sincronización de ejecución entre subsistemas y la correcta inicialización de parámetros de inicio para cada subsistema. Aqui se utiliza el subsistema de comunicaciones para lo referente a determinar en que anfitrión se ejecutan los procesos Cliente y donde el proceso Servidor.

Este subsistema no es un objeto de la jerarquia de objetos del sistema de software propiamente dicho pero representa al administrador del sistema de software ya que controla de cierta manera donde y cuando se ejecutarán los otros subsistemas.

4.4.4 Implementación Subsistema Simulador

Los diagramas de flujo correspondientes al subsistema Simulador se presentan en las figuras 4.17, 4.18, 4.19 y 4.20. El módulo Planta se compone de las siguientes clases:

INTEGRADOR

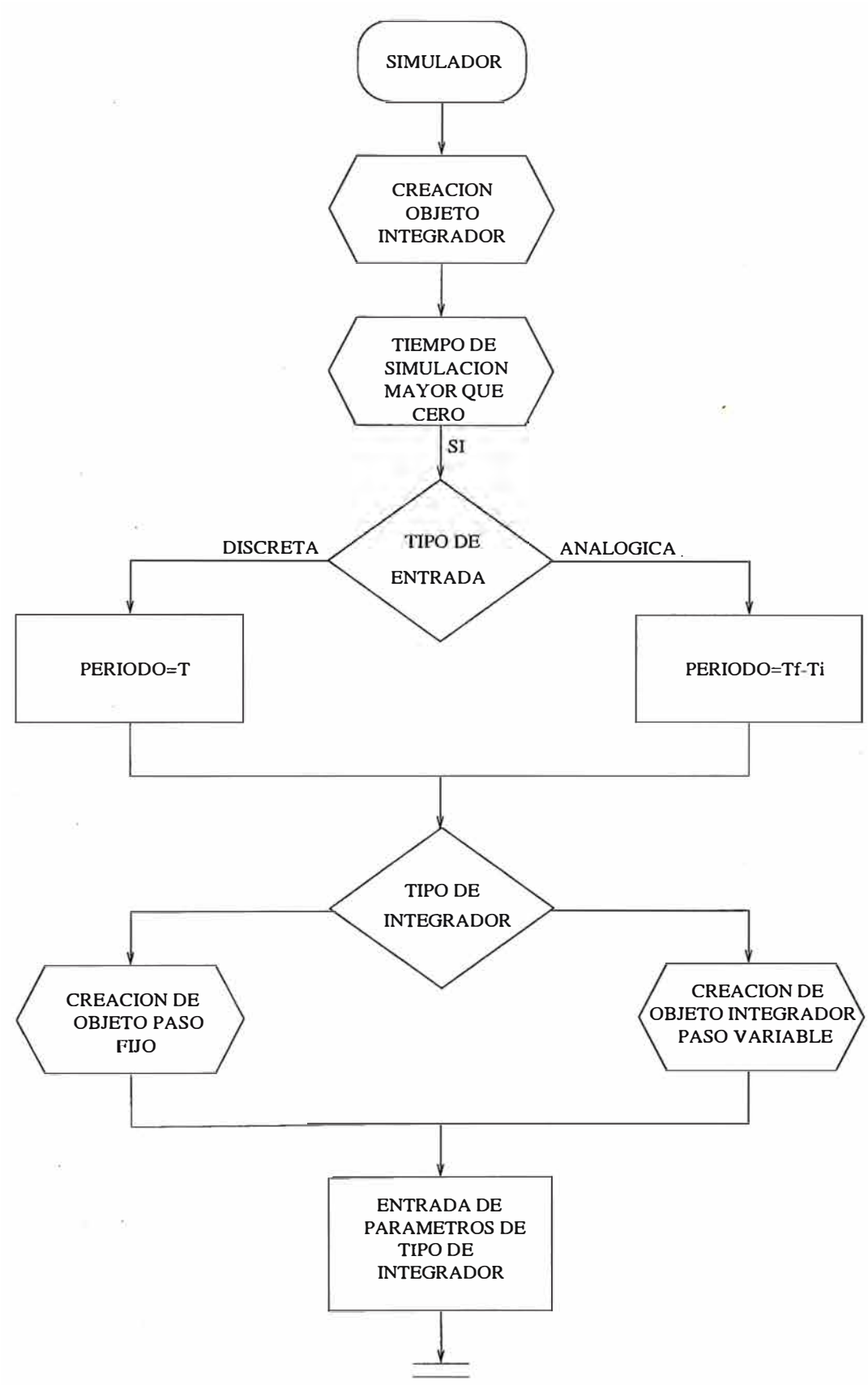

Figura 4.17: Diagrama de Flujo de Subsistema Simulador

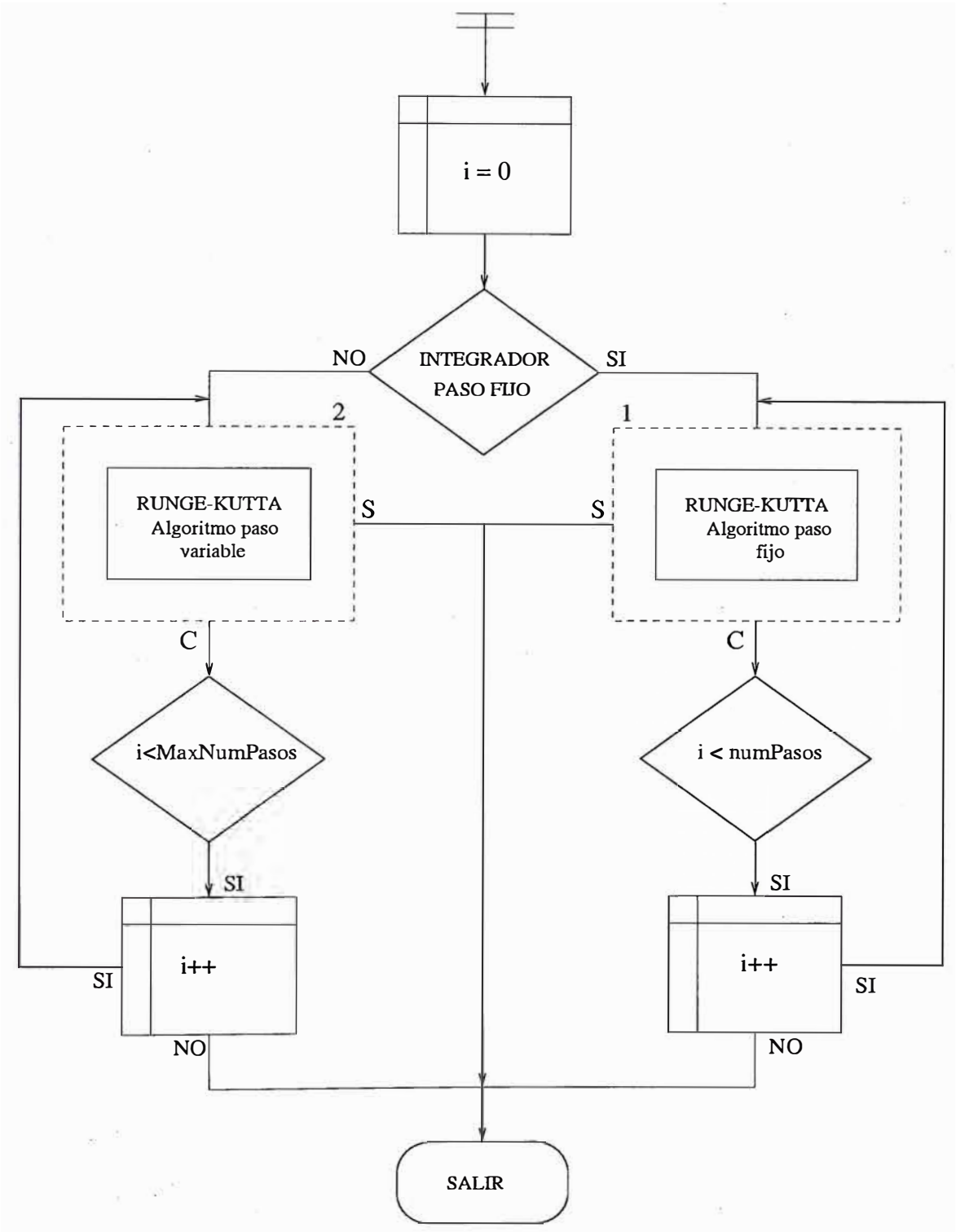

Figura 4.18: Diagrama de Flujo de Subsistema Simulador

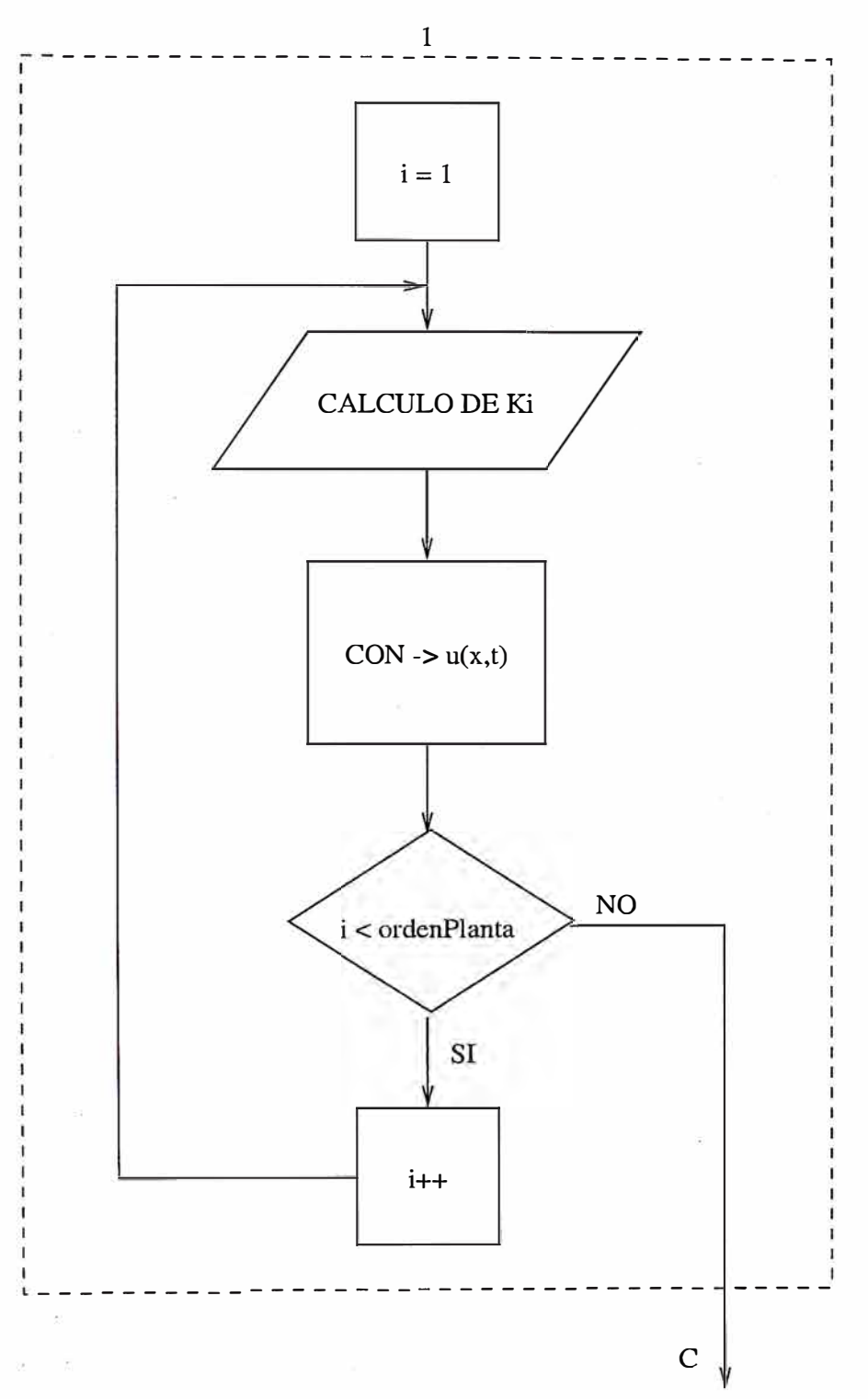

Figura 4.19: Diagrama de Flujo de Subsistema Simulador

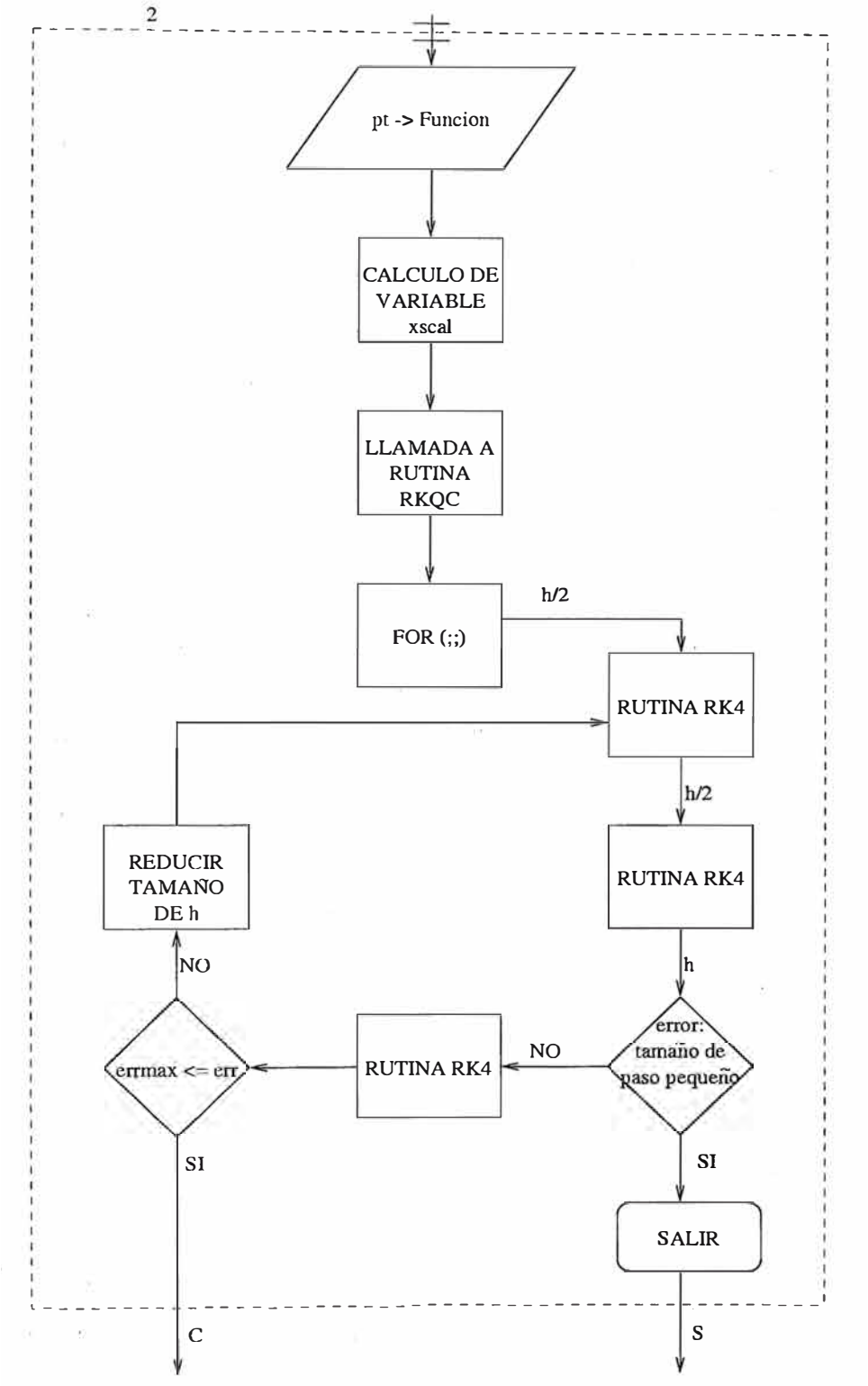

Figura 4.20: Diagrama de Flujo de Subsistema Simulador

Clase Plantas

Descripción de la Clase Planta

DECLARADO EN planta.h

DESCRIPCION DE LA CLASE

La clase `Planta` es una clase base diseñada para manejar los datos miembro comunes a todas las plantas, las que se definen en clases derivadas a esta; cada cual con su propio modelo matemático por medio de una *función virtual pura.*

DATOS MIEMBRO

TIPO DE ACCESO	TIPO	NOMBRE
protected	int	orden
		numParam
		numEntradas
		numSalidas
	Vector	par, aux

DESCRIPCION DE DATOS MIEMBRO

orden	Representa el orden de la planta.
numParam	Número de parametros de la planta.
numEntradas	Número de entradas a la planta.
numSalidas	Número de salidas de la planta.
par	Vector de parametros de la planta.
aux	Vector de valores auxiliares de la planta.

FUNCIONES MIEMBRO

TIPO DE ACCESO	NOMBRE
public	o getOrden
	o getNumParam
	o getNumEntradas
	o getNumSalidas
	o setParametros
	o Funcion
	o salida
protected	o calcularAuxiliares ()

DESCRIPCION DE FUNCIONES MIEMBRO

- **setParametros** :
  - void setParametros (Vector par)

  Función que setea número de parametros de la planta.

- **getOrden** :
  - int getOrden ()

  Retorna el orden de la planta.

- **getNumParam** :
  - int getNumParam ()

  Retorna el número de parametros de la planta.

- **getNumEntradas**
  - int getNumEntradas ()

  Retorna el número de entradas de la planta.

- **getNumSalidas**
  - int getNumSalidas ()

  Retorna el número de salidas de la planta.

- **Funcion** :
  - virtual Vector Funcion (double ,Vector ,Vector) = 0

  Virtual. Se utiliza para definir la función matemática que modela la Planta, en función de la variable de estado x, la variable de entrada u, el tiempo t, los parametros de la planta y los parametros auxiliares.

- **salida**
  - virtual Vector salida (double ,Vector) = 0

  Función virtual que debe retornar el vector salida de la planta.

- **calcularAuxiliares**
  - virtual void calcularAuxiliares () = 0

  Virtual. Calcula los parametros auxiliares necesarios para el modelo matemático de la planta.

Descripción de la Clase PIF

DECLARADO EN pif.h
DESCRIPCION DE LA CLASE

La clase PIF es una clase derivada de la clase Planta que define las instancias comunes del modelo matemático del péndulo invertido, en sus dos casos particulares (péndulo

invertido de bola controlado por fuerza y péndulo invertido de varilla controlado por fuerza).

FUNCIONES MIEMBRO

TIPO DE ACCESO	NOMBRE
public	o Funcion
	o salida

DESCRIPCION DE FUNCIONES MIEMBRO

- **Funcion** :
  - o Vector Funcion (double t, Vector x, Vector u)

    Retorna un vector con las derivadas de estado obtenidas de las ecuaciones del péndulo.

- **salida**
  - o Vector salida (double t, Vector x)

    Retorna el vector salida de la planta.

Descripción de la Clase PIBF

DECLARADO EN pibf.h

DESCRIPCION DE LA CLASE

La clase `PIBF` es una clase derivada de la clase `PIF` donde se definen los parametros y valores auxiliares necesarios para el modelamiento matemático del sistema de *Péndulo Invertido de Bola Controlado por Fuerza.*

FUNCIONES MIEMBRO

public	o PIBF
	o calcularAuxiliares

DESCRIPCION DE FUNCIONES MIEMBRO

- **PIBF** :
  - o PIBF ()

    Constructor. Inicializa los datos del péndulo (sistema PIBF) tal como el número de parametros, las dimensiones de la planta, y los valores de los parametros.

- **calcularAuxiliares**
  - o void calcularAuxiliares ()

    Calcula valores auxiliares para las ecuaciones del péndulo definidas en Funcion.

Descripción de la Clase PIVF

DECLARADO EN pivf.h
DESCRIPCION DE LA CLASE

La clase `PIVF` es una clase derivada de la clase `PIVF` que define los parametros auxiliares necesarios para el modelo matemático del sistema de *Péndulo Invertido de Varilla Controlado por Fuerza.*

FUNCIONES MIEMBRO

public
- PIVF
- calcularAuxiliares

DESCRIPCION DE FUNCIONES MIEMBRO

- **PIVF** :
  - PIVF ()

  Constructor. Inicializa los datos del péndulo (sistema PIVF) : número de parametros y valores de cada uno, dimensiones de la planta.

- **calcularAuxiliares**
  - void calcularAuxiliares ()

  Calcula valores auxiliares para las ecuaciones del péndulo definidas en Funcion.

Descripción de la Clase TanquesConectados

DECLARADO EN tanquesConectados.h
DESCRIPCION DE LA CLASE

La clase `TanquesConectados` es clase derivada de la clase `Planta` que define las funciones que describen el modelo matemático del caso de dos *Tanques conectados* .

FUNCIONES MIEMBRO

TIPO DE ACCESO	NOMBRE
public	o TanquesConectados
	o Funcion
	o salida
	o calcularAuxiliares

DESCRIPCION DE FUNCIONES MIEMBRO

- **TanquesConectados**
  - TanquesConectados ()

  Constructor. Inicializa los datos de los Tanques Conectados, como el número de parametros, valores de cada cual y dimensiones de la planta.

- **Funcion** :
  - Vector Funcion (double t, Vector x, Vector u)

  Retorna un vector con las derivadas de estado obtenidas de las ecuaciones del modelo de los tanques.

- **salida** :
  - Vector salida (double t, Vector x)

  Retorna el vector salida de la planta.

- **calcularAuxiliares** :
  - void calcularAuxiliares ()

  Calcula valores auxiliares para las ecuaciones de los tanques definidas en Funcion.

Descripción de la Clase TanquesSeparados

DECLARADO EN tanquesSeparados.h
DESCRIPCION DE LA CLASE

La clase `TanquesSeparados` es clase derivada de la clase `Planta` que define las funciones que describen el modelo matemático del caso de dos *tanques separados.*

FUNCIONES MIEMBRO

TIPO DE ACCESO	NOMBRE
public	o TanquesSeparados
	o Funcion
	o salida
	o calcularAuxiliares

DESCRIPCION DE FUNCIONES MIEMBRO

- **TanquesSeparados** :
  - TanquesSeparados ()

  Constructor. Inicializa los datos de los Tanques Separados, como el número de parametros, valores de cada cual y dimensiones de la planta.

- **Funcion** :
  - Funcion (double t, Vector x, Vector u)

Retorna un vector con las derivadas de estado obtenidas de las ecuaciones del modelo de los tanques.

- **calcularAuxiliares**
  - void calcularAuxiliares ()

  Calcula valores auxiliares para las ecuaciones de los tanques definidas en Funcion.

- **salida**
  - Vector salida (double t, Vector x)

  Retorna el vector salida de la planta.

Asimismo el módulo Integrador esta conformado por las siguientes clases:

Clase Integrador
Descripción de la Clase Integrador

DECLARADO EN integrador.h
DESCRIPCION DE LA CLASE

La clase `Integrador` es una clase base a partir de la cual se define los métodos de integracion utilizando el *algoritmo de RungeKutta* para la solución de las ecuaciones diferenciales ordinarias de los modelos matemáticos de las Plantas.
Esta clase se diseño para manejar los datos miembro comunes a todo tipo de integrador, ademas define las funciones interface para la elección de los parametros de integración, y se declara virtuales las funciones de integracion propiamente dichas.

DATOS MIEMBRO

TIPO DE ACCESO	TIPO	NOMBRE
public	double	*bufferT
		*bufferX
		*bufferY
		*bufferU
protected	int	numParam, n, m, r
	double	tiempoInicial
		tiempoFinal
		t
	Vector	parametros, x, u, dxdt, y
	Planta	*planta

DESCRIPCION DE DATOS MIEMBRO

*bufferT	Buffer que almacena los valores del tiempo.
*bufferX	Almacena los valores del vector de estado.
*bufferY	Almacena los valores del vector de salida.
*bufferU	Almacena los valores del vector de control.
numParam	Número de parametros del integrador.
n	Dimensión de la planta.
m	Dimensión de la Entrada.
r	Dimensión de la salida.
tiempoInicial	Magnitud del tiempo inicial.
tiempoFinal	Magnitud del tiempo final.
t	Magnitud del tiempo actual.
parametros	Vector de parametros del integrador.
x	Vector de Estado.
u	Vector de Control.
dxdt	Derivadas de estado.
y	Vector salida.
*planta	Apuntador al objeto Planta.

FUNCIONES MIEMBRO

Tipo de Acceso	public
	NOMBRE
Funciones Interface	o setParametros
	o setTiempoSimulacion
	o setX
	o setU
	o setPlanta
	o getX
	o getNumParametros
Funciones Virtuales	o rk4
	o integrar
	o auxiliar
	o setNumPasos
	o getPasos

DESCRIPCION DE FUNCIONES MIEMBRO

- **setParametros**
  - void setParametros (Vector par)

  Asigna el vector con los parametros del integrador al vector parametros.

- **setTiempoSimulacion** :
  - void setTiempoSimulacion (double ti ,double tf)

  Asigna el tiempo inicial y final de simulación.

- **setPlanta**
  - void setPlanta (Planta *p)

  Selecciona el tipo de planta a usar.

- **setX** :
  - void setX (Vector newx)

  Asigna el vector de estado al vector x.

- **setU**
  - void setU (Vector newu)

  Asigna el vector de control al vector u.

- **setNumPasos**
  - virtual void setNumPasos (int) = 0

  Virtual. Asigna el número de pasos deseado para integración.

- **getX** :
  - Vector getX ()

  Retorna el vector de estado x.

- **getNumParametros** :
  - int getNumParametros ()

  Retorna el número de parametros de la planta.

- **getPasos ()**
  - int getPasos ()=0

  Virtual. Retorna el número de pasos de integración.

- **rk4** :
  - virtual void rk4 ()=0

  Virtual. Resuelve las Ecuaciones Diferenciales Ordinarias de la Planta mediante el algoritmo de RungeKutta de grado cuarto.

- **integrar**
  - virtual void integrar ()=0

  Virtual. Define el manejador de la rutina rk4 para la integración en un intervalo determinado.

- **auxiliar** :
  - virtual void auxiliar (int, int)=0

  Virtual. Define algunos valores auxiliares para el tipo de integrador.

Descripción de la Clase IntegradorAnalogico

DECLARADO EN integrAnalogico.h
DESCRIPCION DE LA CLASE

La clase `IntegradorAnalogico` es una clase derivada de la clase `Integrador` que define los métodos de integración para una *entrada analogica* en la planta.

DATOS MIEMBRO

TIPO DE ACCESO	TIPO	NOMBRE
protected	double	h
	int	output, input
	VectorCode	X, U, Y
	FloatCode	T

DESCRIPCION DE DATOS MIEMBRO

h	Tamaño de paso de integración.
output	Descriptor de salida.
input	Descriptor de entrada.
X, U, Y	Vectores para codificar y decodificar vectores.
T	Para codificar y decodificar floats.

FUNCIONES MIEMBRO

public
- rk4

DESCRIPCION DE FUNCIONES MIEMBRO

- **rk4** :
  - void rk4 ()

  Resuelve las *ODE's* de la Planta asumiendo una entrada analogica.

Descripción de la Clase IntegradorDiscreto

DECLARADO EN integrDiscreto.h
DESCRIPCION DE LA CLASE

La clase `IntegradorDiscreto` es una clase derivada de `Integrador` que define la función de integración para una entrada discreta.

DATOS MIEMBRO

TIPO DE ACCESO	TIPO	NOMBRE
protected	double	h

DESCRIPCION DE DATOS MIEMBRO

h	Tamaño de paso de integración.

FUNCIONES MIEMBRO

public
- rk4

DESCRIPCION DE FUNCIONES MIEMBRO

- **rk4** :
  - void rk4 ()

  Resuelve las *ODE's* de la planta para una entrada discreta.

Descripción de la Clase PasoFijoAnalogico

DECLARADO EN pasoFijoAnalogico.h

DESCRIPCION DE LA CLASE

La clase `PasoFijoAnalogico` es una clase derivada de `IntegradorAnalogico` que define el manejador de RungeKutta con paso fijo, ademas de valores auxiliares.

DATOS MIEMBRO

TIPO DE ACCESO	TIPO	NOMBRE
private	int	nsteps

DESCRIPCION DE DATOS MIEMBRO

nsteps	Número de pasos de integración.

FUNCIONES MIEMBRO

public
- PasoFijoAnalogico
- integrar
- auxiliar
- getPasos
- setNumPasos

DESCRIPCION DE FUNCIONES MIEMBRO

- **PasoFijoAnalogico**
  - PasoFijoAnalogico ()

  Constructor. Asigna el número de parametros del integrador.

- **integrar**
  - void integrar ()

  Manejador de rk4 con tamaño de paso fijo para entrada analogica.

- **auxiliar** :
  - void auxiliar (int in, int out)

  Asigna los ordenes correspondientes a los parametros de integrador de acuerdo al tipo de planta elegido.

Descripción de la Clase PasoFijoDiscreto

DECLARADO EN pasoFijoDiscreto.h
DESCRIPCION DE LA CLASE

La clase `PasoFijoDiscreto` es una clase derivada de `IntegradorDiscreto` que define el manejador de RungeKutta con paso fijo para entrada discreta.

DATOS MIEMBRO

TIPO DE ACCESO	TIPO	NOMBRE
private	int	nsteps

DESCRIPCION DE DATOS MIEMBRO

nsteps	Número de pasos de integración.

FUNCIONES MIEMBRO

public
- PasoFijoDiscreto
- integrar
- auxiliar
- getPasos
- setNumPasos

DESCRIPCION DE FUNCIONES MIEMBRO

- **PasoFijoDiscreto** :
  - PasoFijoDiscreto ()

  Constructor. Asigna el número de parametros del integrador

- **integrar** :
  - void integrar ()

  Manejador de rk4 con tamaño de paso fijo para entrada discreta.

- **auxiliar**
  - void auxiliar (int in, int out)

  Asigna los ordenes correspondientes a los parametros de integrador de acuerdo al tipo de planta elegido.

Descripción de la Clase PasoVariableAnalogico

DECLARADO EN pasoVariableAnalogico.h

DESCRIPCION DE LA CLASE

La clase `PasoVariableAnalogico` es una clase derivada de `IntegradorAnalogico` que define el algoritmo de RungeKutta con paso variable.

DATOS MIEMBRO

TIPO DE ACCESO	TIPO	NOMBRE
public	int	steps
protected	int	MAXSTP
	double	hi, hmin, eps
private	double	hnext
	Vector	xscal,xtemp,xtmp,xsav,dxsav

DESCRIPCION DE DATOS MIEMBRO

steps	Número de pasos de integración realizados.
MAXSTP	Maximo número de pasos permitido.
hi	Tamaño de paso tentativo.
hmin	Minimo tamaño de paso permitido.
eps	Exactitud requerida para calidad de paso.
hnext	Estimación de tamaño de paso siguiente.
xscal	Calculo de un vector escala para comparación.
xtemp, xtmp	Vectores auxiliares para almacenar x.
xsav, dxsav	Guarda los valores de condiciones iniciales.

FUNCIONES MIEMBRO

public
- PasoVariableAnalogico
- controlCalidadPaso
- integrar
- auxiliar

DESCRIPCION DE FUNCIONES MIEMBRO

- **PasoVariableAnalogico** :
  - PasoVariableAnalogico ()

  Constructor. Asigna el número de parametros del integrador.

- **controlCalidadPaso** :
  - void controlCalidadPaso ()

  Función que controla la calidad del tamaño de paso necesaria para la rutina de paso variable.

- **integrar** :
  - void integrar ()

  Manejador de rk4 y controlCalidadPaso para integración con paso variable.

- **auxiliar** :
  - void auxiliar (int in, int out)

  Asigna los ordenes correspondientes a los parametros de integrador de acuerdo al tipo de planta elegido.

Descripción de la Clase PasoVariableDiscreto

DECLARADO EN pasoVariableDiscreto.h

DESCRIPCION DE LA CLASE

La clase `PasoVariableDiscreto` es una clase derivada de `IntegradorDiscreto` que define el manejador de RungeKutta con paso variable para entrada discreta.

DATOS MIEMBRO

TIPO DE ACCESO	TIPO	NOMBRE
public	int	steps
protected	int	MAXSTP
	double	hi, hmin, eps
private	double	hnext
	Vector	xscal,xtemp,xtmp,xsav,dxsav

DESCRIPCION DE DATOS MIEMBRO

steps	Número de pasos de integración realizados.
MAXSTP	Maximo número de pasos permitidos.
hi	Tamaño de paso tentativo.
hmin	Minimo tamaño de paso permitido.

eps	Exactitud requerida.
hnext	Estimación de tamaño de paso siguiente.
xscal	Calculo de un vector escala de comparación.
xtemp, xtmp	Vectores auxiliares para almacenar x.
xsav, dxsav	Guarda los valores de condiciones iniciales.

FUNCIONES MIEMBRO

public
- PasoVariableDiscreto
- controlCalidadPaso
- integrar
- auxiliar

DESCRIPCION DE FUNCIONES MIEMBRO

- **PasoVariableDiscreto** :
  - PasoVariableDiscreto ()

  Constructor. Asigna el número de parametros del integrador.

- **controlCalidadPaso**
  - void controlCalidadPaso ()

  Controla la calidad del tamaño de paso necesaria para la rutina de paso variable.

- **integrar** :
  - void integrar ()

  Manejador de rk4 y controlCalidadPaso para integración con paso variable.

- **auxiliar**
  - void auxiliar ()

  Asigna los ordenes correspondientes a los parametros de integrador de acuerdo al tipo de planta elegido.

4.4.5 Implementación Subsistema Controlador

El diagrama de flujo del subsistema Controlador se presenta en la figura 4.21. El módulo Controlador se divide en las siguientes clases:
La clase Controlador que es una clase base para definir controladores para las plantas. La clase Realimentación de Estado que es una clase derivada de la clase base Controlador y que define el controlador Realimentado de Estado.

La parte de comunicaciones que involucra el subsistema esta dada por el programa Cliente/Servidor antes ya mencionado que utiliza sockets como medio de comunicación

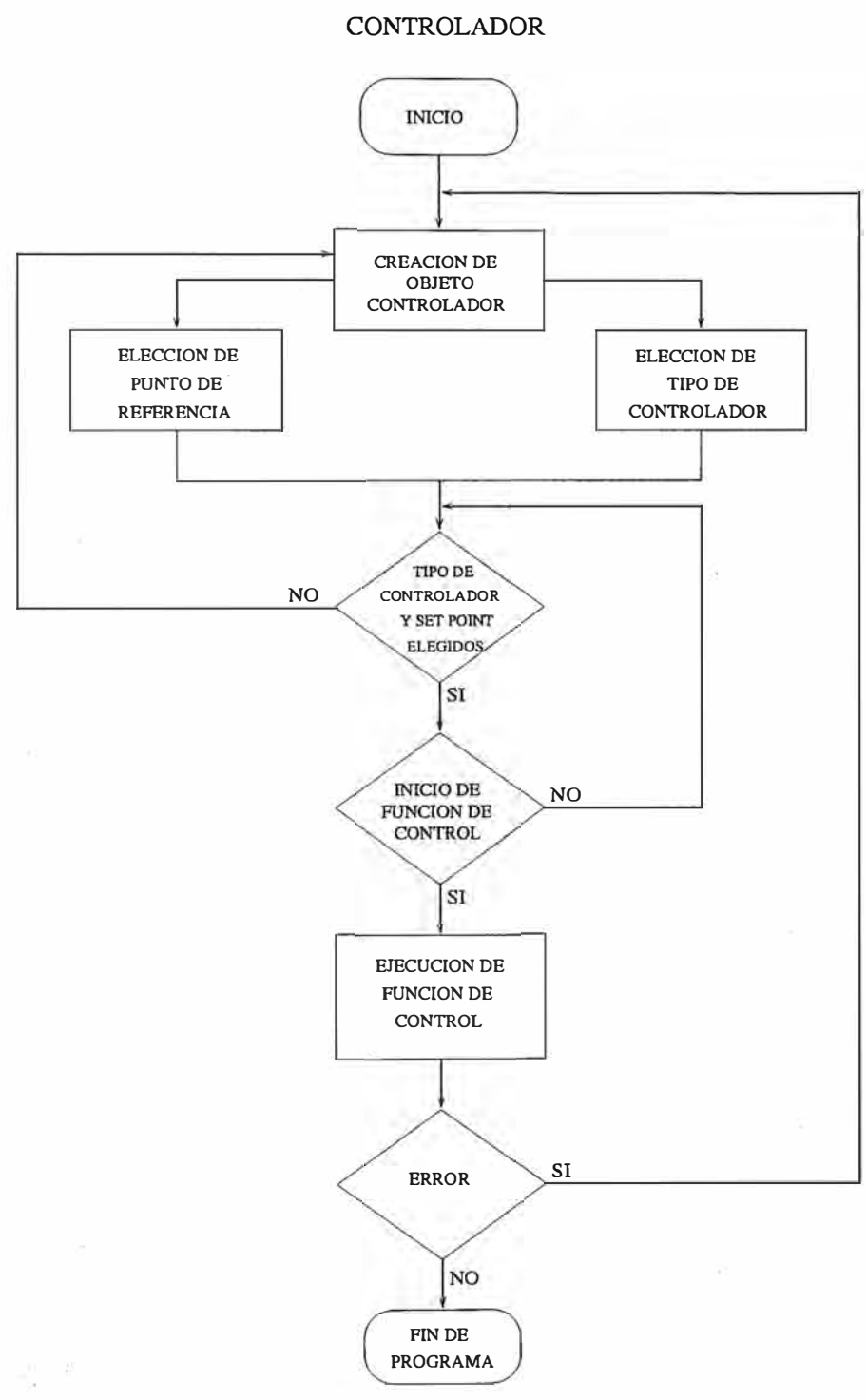

Figura 4.21: Diagrama de Flujo de Subsistema Controlador

y que permite la codificación y decodificación de información que intercambia este subsistema con el subsistema Simulador a través del módulo de Integración.

4.4.6 Implementación Subsistema Visualizador

El proceso de visualización [24] es una secuencia de transformaciones que convierten un conjunto de datos en algo que se puede graficar. La manipulación de datos convierte un conjunto de datos a una conveniente forma para posteriores operaciones de visualización, tales como interpolación y enrejado. Mientras que el mapeo de visualización define una técnica de visualización abstracta para establecer un conjunto de correspondencia entre los datos manipulados y las primitivas de visualización, como parametros posicionales, color, texto, y animación. El objetivo primordial del mapeo de visualización es identificar un conjunto de primitivas que puedan manejar adecuadamente la información contenida en los datos.

En este subsistema la implementación esta formada por el programa que implementa todas las rutinas gráficas de Visualización y que utiliza la aplicación Gnuplot como destino de la información a dibujar. La conexión con Gnuplot se establece mediante un pipe, que implementado bajo Unix representa un punto de comunicación unidireccional. De esta manera cada pipe abierto hacia la aplicación Gnuplot representa una petición de la aplicación por parte del usuario, lo que permite trabajar cada pipe hacia Gnuplot de manera independiente.

4.4.7 Implementación de Subsistema Comunicaciones

Se describen las clases Servidor y Cliente que han sido implementadas para crear aplicaciones cliente-servidor. Las clases Servidor y Cliente hacen posible la comunicación entre dos o más procesos que pertenecen a un mismo sistema o a diferentes sistemas. Se usa el mecanismo de comunicaciones ofrecido por *UNIX*: los *sockets* ,con el protocolo `TCP\IP`. Estas clases fueron desarrolladas con la finalidad de ocultar la complejidad de la utilización directa de los *sockets*.

Clase Servidor

DECLARADO EN: Servidor4.h
DESCRIPCION DE LA CLASE:

Esta clase implementa un punto de comunicación para cualquier programa servidor. Una vez conectado a un programa cliente, el programa servidor podrá enviar y recibir información del cliente (primitivas read(), write(), send(), recv()), utilizando un descriptor que referencia dicho punto de comunicación.

DATOS MIEMBRO:	TIPO	NOMBRE
	struct sockaddr_in	addr;
	int	puerto;
	int	nClientes
	int	descriptor
	int	descripComun

addr	(privada)	direccion internet del socket del Servidor
puerto	(privada)	puerto utilizado para la comunicacion
nClientes	(privada)	numero de clientes que se aceptaran (max. 5)
descriptor	(privada)	descriptor del socket del Servidor
descripcomun	(publica)	descriptor utilizado para la comunicacion

FUNCIONES MIEMBRO:

- Servidor
- ~Servidor
- aceptar
- setServ

DESCRIPCION DE FUNCIONES MIEMBRO:

- **Servidor**

  - Servidor (int port,int numeroCli)

  Constructor. Crea un punto de comunicacion por el puerto `port` (mayor que 10000 de preferencia, porque los primeros puertos son reservados, el minimo es 64000) y podrá conectarse con un numero de clientes igual a numeroCli (el maximo es 5).

- **Servidor**

  - Servidor()

  Destructor. Hace un llamado a la primitiva `close()` para liberar el socket creado con el constructor.

- **aceptar**

  - int aceptar ()

  Cuando existe un programa cliente conectado al puerto usado por el Servidor, esta función acepta a dicho cliente y copia en `descripComun` el descriptor que se usara para la comunicacion (si hay exito en la coneccion) y retorna al valor 0. En caso de fallas retorna el valor -1.

- **setServ**

  - void setServ (int a, int b)

  Setea los valores de los datos miembro `puerto` y `numClientes` con a y b respectivamente y enlaza el socket del Servidor al puerto de número `puerto`.

Clase Cliente

DECLARADO EN: Cliente4.h

DESCRIPCION DE LA CLASE:

Esta clase implementa un punto de comunicación para cualquier programa cliente. Una vez conectado a un programa servidor, el programa servidor podrá enviar y recibir información del cliente (primitivas read(), write(), send(), recv()), utilizando el descriptor que referencia dicho punto de comunicación.

DATOS MIEMBRO:	TIPO	NOMBRE
	struct sockaddr_in	serverAddr;
	int	puerto;
	char*	serverHost;
	int	descriptor

serverAddr	(privada)	direccion internet del socket del Servidor al cual se conectara el Cliente
puerto	(privada)	puerto utilizado para la comunicacion
serverHost	(privada)	el nombre del host donde esta el Servidor
int	(publica)	descriptor del socket del Cliente

FUNCIONES MIEMBRO:

- Cliente
- ~Cliente
- conectar
- setClient

DESCRIPCION DE FUNCIONES MIEMBRO:

- **Cliente**

  - Cliente (int port, char *host)

  Constructor. Se crea un punto de comunicación en el puerto `port`, a través del cual nos conectaremos a un Servidor que está corriendo en el sistema de nombre `host`.

- **Cliente**

  - Cliente()

  Destructor. Libera el socket creado por el constructor mediante una llamada a la primitiva `close()`.

- **conectar**

  - int conectar (int wait)

  Permite la conección a un Servidor corriendo en `serverHost` (seteada por el constructor). Si wait es 1 se espera hasta que el Servidor esté conectado al puerto. Si wait es 0 , se intentará una sola vez la comunicación con el Servidor. En caso de conección se retorna el valor de 0. Si no se logra la conección se retorna el valor de -1.

- **setClient**

  - void setClient (int port, char *host)

  Setea los datos miembro `puerto` y `serverHost` con los valores de `port` y `host` respectivamente.

4.4.8 Integración de Subsistemas de Software

La integración de los subsistemas de software se da como última etapa de la parte de diseño e implementación, con el propósito de lograr una versión completa del sistema de software funcionando cunmpliendo todos los requisitos bajo los cuales fue diseñado para garantizar el análisis realizado.

Esta tarea se realizó por partes, lo primero que se realizó fue verificar cada uno de los subsistemas en forma individual para detectar posibles errores en la implementación de alguno de ellos. Posteriormente se realizó la implementación del código necesario para lograr que los subsistemas lograran tener un funcionamiento preestablecido, además de permitir la ejecución de los subsistemas en anfitriones diferentes de tal manera que se logre el funcionamiento distribuido del sistema. La mayor parte del código de comunicación reside en el subsistema Interface Usuario y el código que involucra a cada subsistema es similar. Asi tenemos por ejemplo que la codificación al momento de elegir en que anfitrión se ejecutará uno de los subsistemas será tal como:

```
switch(remoteHost) {
    case 'v':
      cli.setCliente(SOCKET_IUP_SIMULADOR, "viru");
      cli.conectar(1);
      break;
```

```
    case 'w':
      cli.setCliente(SOCKET_IUP_SIMULADOR, "wari");
      cli.conectar(1);
      break;
}
```

Mientras que en un subsistema se implementará la codificación necesaria para establecer la comunicación y permitir la inicialización de los parámetros correspondientes al subsistema para su funcionamiento completo.

```
/* Iniciar servidor para conectarse con IUP */
  Servidor ser(SOCKET_IUP_CONTROLADOR);
  ser.aceptar();

/* Obtener numero de entradas , salidas y host de IUP */
  read(ser.descripComun, &entradas, 1);
  read(ser.descripComun, &estados, 1);
  read(ser.descripComun, &hostIUP, 1);
```

Posterior a la codificación para permitir la operación del sistema como una sóla unidad operativa se realizó una serie de pruebas para asegurar el cumplimiento de los requisitos que se especificaron en el software inicialmente. Estas pruebas no estuvieron libres de errores por lo que la integración permitió resolver problemas no detectados durante la prueba individual de los subsistemas. Más adelante en el próximo capitulo se realizará un análisis más detallado de las pruebas de integración que se realizarón.

CAPÍTULO V
EVALUACIÓN Y VERIFICACIÓN DE SISTEMA DE SOFTWARE

5.1 Evaluación del proceso de Gestión del Proyecto de Software

En la tabla 5.1 observamos el resultado de las lineas de código que resultaron en cada uno de los subsistemas del SCD_Avanzado y se le compara con las lineas de código estimadas inicialmente durante la fase de gestión del software.

El elemento en discusión, lineas de código es un factor muy sensible en los cambios que se realizaron durante la etapa de desarrollo del software, tomando en cuenta que para las interfaces gráficas se utilizó el lenguaje de programación Xview cuya particularidad es la regular cantidad de lineas de código necesarias para la escritura de un programa sencillo.

De los resultados de la tabla tenemos que los subsistemas Interface Usuario y el de Comunicaciones tiene el factor LDC más cercano al valor esperado durante la etapa de gestión del proyecto. Esto porque primero el subsistema de Comunicaciones no tiene una interface al usuario como los demás subsistemas y respecto al subsistema Interface Usuario porque la interface que presenta es pequeña comparada con la de los otros subsistemas y es factible con mayor facilidad estimar el número de lineas de código. Respecto al resto de los subsistemas notamos que se realizó una buena aproximación en el subsistema Visualizador, pues el número de lineas de código resultante está por debajo del cálculo más pesimista que se realizó. En el subsistema Controlador y Simulador el número de lineas de código si estuvo por encima del cálculo más pesimista, la causa principal es que en ambos subsistemas las interfaces de usuario contienen código no perteneciente a Xview que no se consideró como medida importante en la gestión del

	Resultantes	Esperada
IUP	1770	1600
Controlador	3100	1050
Simulador	3720	1267
Visualizador	12950	11834
Comunicaciones	1046	734
Total	22586	16485

Tabla 5.1: Resultado de LDC para SCD_Avanzado

proyecto.

Como resultado general el número de linea de código que se estimó para todos los subsistemas está dentro de un rango aceptable tomando en cuenta que el número de linea de código totales del subsistema no superan las estimaciones más pesimistas que se realizarón. La estimación de LDC para nuestro proyecto de software resulto siendo una buena alternativa de estimación para medir la dimensión del sistema de software.

En lo que se refiere al análisis de riesgos realizado anteriormente mencionamos que dejaron de ser una posibilidad y se convirtieron en riesgos reales, afectando el alejamiento de algunos participantes en el proyecto inicialmente para luego tener mayor claridad en los pasos a seguir en el proyecto (si se abandonaba el proyecto definitivamente o se continuaba hasta finalizarlo). Fueron importantes en esta etapa las reuniones semanales del equipo de trabajo, pues esto contribuyo a que se tuviera una visión general del trabajo del grupo para prevenir justamente casos como los que se presentaron después de integrantes que no concluyeron toda la ejecución del proyecto. De esta manera cuando alguien se alejaba del proyecto, su trabajo era conocido por el resto del grupo y facilitaba un poco el hecho de asumir el trabajo pendiente.

5.2 Evaluación de recursos humanos, recursos de software y recursos de hardware

Con respecto a la estimación del personal necesario para participar en el proyecto realizada durante la etapa de gestión del proyecto hay que mencionar que a pesar que el número de personas inicialmente pensadas para participar en el proyecto disminuyó durante la ejecución del mismo, pues de cinco personas pensadas inicialmente sólo dos culminaron el proyecto; a pesar de ello los tiempos no se prolongaron demasiado tomando en cuenta el impacto que se pudo producir por el alejamiento de más de la mitad del equipo inicial y considerando factores como la falta de experiencia del personal, el tiempo disponible para el proyecto considerandoque los participantes en su mayoria eran estudiantes.

Asi de un tiempo de duración estimado del proyecto de 14.7 meses, el tiempo de ejecución real del mismo resultó ser de 19 meses considerando durante esos meses jornadas de 20 horas/semana. Como conclusión el personal que participó en el proyecto resulto el adecuado en lo que a preparación y capacidad de trabajo se refiere, tomando en cuenta los conocimientos que se adquirieron durante la ejecución del proyecto fue beneficioso además para la preparación profesional de los estudiantes participantes.

Respecto al software utilizado, el lenguaje de programación C^{++} resulto ser una buena elección por la facilidad de lectura del código implementado con este lenguaje,

además de permitirnos obtener facilidades que proporciona la programación orientada a objetos. Su semejanza al lenguaje C hizo de que no existieran inconvenientes cuando se programo con objetos en interfaces de usuario creadas con Xview.

Además el lenguaje C^{++} nos permitio construir el sistema con una jerarquia de objeto del sistema de software que es flexible y que brinda gran cantidad de código reutilizable para otras aplicaciones. Con relación al lenguaje Xview se comparo con otra aplicación con facilidades para la creación de interfaces de usuario implementando código C^{++} como es la aplicación Interviews. Como resultado de la comparación se tuvo que con el lenguaje Xview se implementa programas mucho más largos que con Interviews, además de que realizar cambios en Interviews resulta ser más simple que en Xview, pues Interviews es un generador de código C^{++}. La razón por la que aun se opto por Xview es que a diferencia de Interviews Xview está estandarizado para muchas plataformas mientras que Interviews no, por ello Xview cumple con mayor amplitud nuestros deseos de lograr un software portable, asumiendo con ello el costo que significa implementar código más engorroso de cierta manera.

El programa Matlab fue de gran ayuda para la etapa de simulación de los procesos ahorrando horas de trabajo para la obtención de los mismos resultados. Respecto a la forma como se transfiere datos entre computadoras con distinta arquitectura se intento utilizar una libreria llamada DTM, lo cual sin embargo no logro concretarse pues se descubrio que era una versión disponible para estaciones de trabajo solamente, lo que involucraba camios si se deseaba como en nuestro caso instalarla en una PC, por lo que se decidio construir una interface de comunicación propia. Algo que se debe mencionar aqui es que los programas fuente del sistema de software se compilaron tanto en PC's como en un equipo Sun Sparc sin tener que modificar los programas fuente, lo que asegura portabilidad del sistema, además de confirmar que el equipo de computo utilizado posee indices de calidad excelentes, tanto por la arquitectura interna como por el software que tienen instalado.

5.3 Verificación de Subsistemas de Software

Cada uno de los subsistemas como etapa previa a la integración del sistema de software fue probado individualmente. Esta prueba no es producto de una etapa final sino que por el contrario se va dando a tráves del desarrollo del sistema de software. Para la verificación individual de los subsistemas se utilizarón varios criterios:

- Las pruebas se realizarón utilizando como parámetros de cada subsistema los valores limite permitidos, buscando con ello ver el comportamiento ante imprevistos.

- Las pruebas buscaran que el programa se ejecute buscando la mayor cantidad de sentencias dentro del programa, es decir se busca que ninguna linea de código quede sin ser probada.

- Se probara si los tiempos de respuesta cambian demasiado cuando el sistema operativo esta ejecutando varias tareas simultaneamente.

Para el subsistema Simulador primero se verificó el ingreso correcto de la configuración del subsistema, si se validaban letras donde sólo se permiten numeros, luego la correcta ejecución de cada una de los botones de la interfase de usuario. Una segunda prueba consistio en configurar los parámetros con sus valores limite como en el caso del proceso de integración donde se probó con pasos de integración nulos y valores mayores que el tiempo de simulación, para verificar el comportamiento del sistema. Una tercera prueba fue probar la simulación con cada una de las plantas disponibles con el S.O. ejecutando varias tareas; aquí se noto una pérdida de desempeño en el subsistema.

Para el subsistema Controlador se hicieron pruebas similares que para el subsistema Simulador en lo referido a validación de argumentos, detección de errores de validación y funcionamiento como en el subsistema anterior. Adicionalmente se realizó una prueba de funcionamiento conjunto con el subsistema Simulador obteniendose resultados satisfactorios. Aquí se comprobo que sucedia cuando uno de los subsistemas funcionaba y el otro no, el punto de comunicación establecido entre ambos subsistemas para la transferencia de datos quedaba en tiempo de espera durante cinco minutos luego de los cuales presentaba un mensaje especificando el detalle de lo ocurrido. No es necesario que un subsistema se ejecute antes que otro, lo improtante es que se establezca la comunicación entre ambos dentro de un tiempo limite establecido.

Para el caso del subsistema Visualizador se realizarón las primeras pruebas de validación de información, luego se paso a las pruebas de integridad de información del subsistema que nos permiten saber si los subsistemas validan la informacion correctamente y si el tratamiento que le dan es el apropiado de acuerdo a lo especificado en la etapa de gestión. Como última prueba se verifico su funcionamiento trabajando con varias ventanas y con varias señales en una ventana verificandose la creación y eliminación de las mismas. En esta parte se hizo necesaria más de una modificación en el subsistema, sobretodo de código de validación.

5.4 Prueba de Integración de Sistema de Software

La prueba de integración del Sistema de Software se produce como paso final del desarrollo del proyecto y tiene como objetivo eliminar nuevos errores que se puedan presentar durante la integración del sistema de software, ademas de ser el último paso

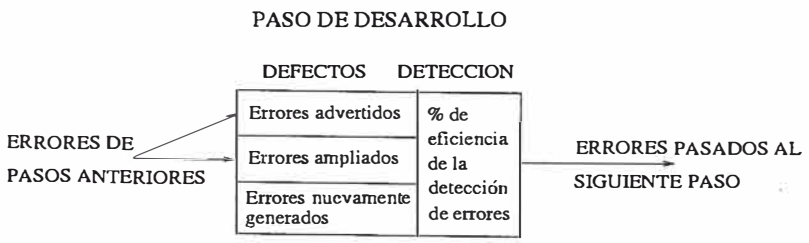

Figura 5.1: Paso de Desarrollo. Prueba de Software

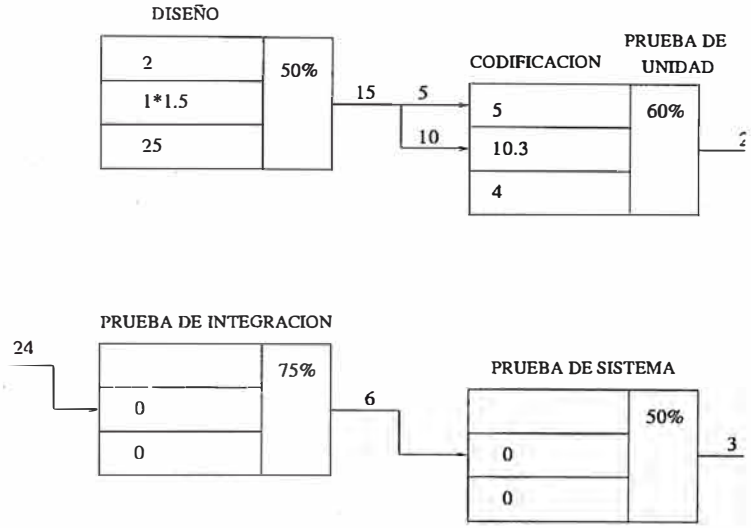

Figura 5.2: Prueba de Software. Subsistema Simulador

luego del cual, si los resultados son positivos se esta en condiciones de mostrar una versión operativa funcionando de acuerdo a las especificaciones de diseño elegidas. Para esta etapa se realizarón dos tipos de pruebas ya realizadas anteriormente para la verificación individual de los subsistemas, estas son la prueba de la caja blanca que nos permite obtener una medida de la complejidad lógica de un diseño y usar esa medida como guia para la definición de un conjunto básico de caminos de ejecución que nos permitiran ejecutar por lo menos una vez la mayor cantidad de sentencias de un programa, y la prueba de la caja negra se centra en probar los requisitos funcionales del sistema de software. Ambas pruebas no son excluyentes una de la otra sino mas bien complementarias.

Los resultados de estas pruebas se resumen bajo la presentación con una sintaxis que se ve en la figura 5.1, donde el factor x es variable de acuerdo a la etapa de desarrollo y la complejidad del software. Los resultados de las pruebas vistas desde cada subsistema del sistema de software bajo el modelo de presentación de existencia de errores anterior se presentan en las figuras 5.2, 5.3 y 5.4

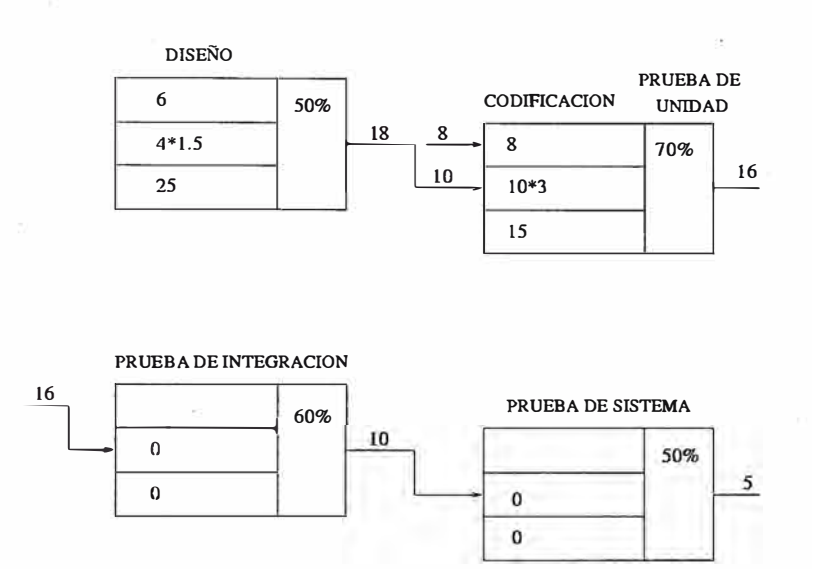

Figura 5.3: Prueba de Software. Subsistema Controlador

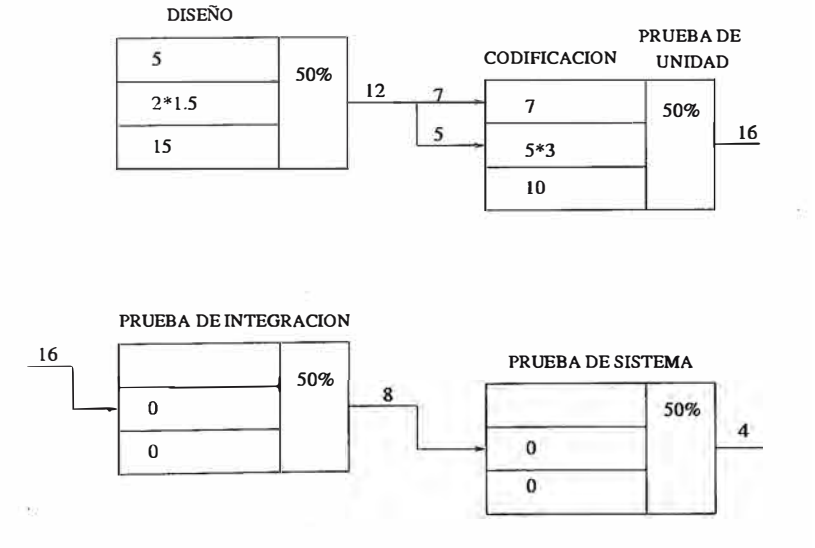

Figura 5.4: Prueba de Software. Subsistema Visualizador

CAPÍTULO VI
RESULTADOS

- Se resolvio el modelo matemático de los procesos Péndulo Invertido y Tanque Doble, implementadose luego un controlador realimentado de estado para ambos procesos.

- Se concluyo que un controlador PID no es aplicable para el caso del péndulo invertido pues no logra estabilizarlo, recomendandose la implementación de un controlador realimentado de estado.

- Se utilizó modelos de estimación y metricas de calidad de software en el proceso de gestión del sistema de software, comprobandose que tienen utilidad para el desarrollo de software, mientras el desarrollo se realize con disciplina y orden.

- Se implementó cuatro subsistemas:

  1. Interface Usuario
  2. Simulador
  3. Controlador
  4. Visualizador

- Los procesos elegidos a controlar son el Péndulo Invertido y el Tanque Doble. Estos fueron implementados como clases derivadas de la clase Planta. La prueba de las clases se realizó con ejemplos de asignación de datos y construcción de objetos; con la finalidad de comprobar los conceptos de POO (Programación Orientada a Objetos).

- Para la implementación del sistema de software se utilizó el lenguaje de programación C^{++}, concluyendose que a pesar de no ser único para implementaciones orientadas a objetos, otorga sin embargo cierta disciplina en su sintaxis y elegancia en el diseño.

- La solución numérica del modelo matemático de los procesos a controlar se implementó en clases derivadas de la clase Integrador. La prueba de los objetos se realizó en conjunto con los objetos de la clase Planta. Para comprobar los resultados se utilizó las simulaciones del modelo obtenidas en **MATLAB**.

- Se diseño las interfaces gráficas de usuario usando XView (lenguaje de programación para objetos gráficos) que se ejecutan bajo ambiente XWindow.

- El funcionamiento de las interfaces fue probado individualmente. Primero verificando el correcto comportamiento de los objetos gráficos que conforman las interfaces. Luego se realizó la prueba con los subsistemas funcionando bajo control de las interfaces; verificandose que las funciones del subsistema correspondientes a cada objeto gráfico funcionen apropiadamente.

- La comunicación entre los subsistemas se realiza a tráves de mensajes del protocolo TCP/IP usando sockets, la interface de comunicación implementada en UNIX. Para ello se implementó un subsistema de comunicaciones bajo el modelo de Cliente/Servidor.

- Se evaluó los recursos utilizados en el proyecto y se hizo una prueba constante de los errores que aparecian en el sistema de software, constituyendose asi la etapa de prueba de unidad e integridad de los subsistemas, no como un paso final sino como un proceso continuo de prueba y corrección.

- Finalmente se realizó la prueba de la aplicación completa en tres computadoras en red: dos PC's con sistema operativo Linux y una Sun SPARClassic con SunOS 5.2. Los resultados obtenidos fueron satisfactorios para el control de los procesos.

CAPÍTULO VII
FUTURO DESARROLLO DEL SISTEMA DE SOFTWARE

Nos dedicaremos a mencionar y explicar algunas ideas de como seguir ampliando el sistema de software SCD_Avanzado, para aumentar sus posibilidades para experimentos con modelos reales, animación gráfica de los procesos simulados o permitir que se puedan agregar nuevos módulos sin tener que recompilar la aplicación completa. Basicamente nos centraremos en tres ideas:

- **Control de procesos reales.**

  Lograr el control de procesos reales a tráves del SCD_Avanzado pasa por varias etapas. La primera de ellas es escribir un programa manejador de dispositivo hardware del S.O. Linux que resulta siendo una colección de subrutinas y datos dentro del kernel y que se constituye en el software de interface a un dispositivo de I/O. De esta manera cuando el kernel reconozca que una acción particular es requerida desde el dispositivo, llama a la rutina apropiada del manejador, que pasa el control desde el proceso usuario a la rutina del manejador. El control es retornado al proceso usuario cuando la rutina del manejador es completada.

  Para nuestro caso se hace necesario escribir un manejador de dispositivo de tipo caracter que nos permita accesar fácil y rapidamente a la memoria del hardware, que puede ser una tarjeta de adquisición de datos.

  Hay que tener en cuenta que un proceso de usuario Linux no puee accesar directamente a la memoria física. El esquema de manejo de memoria supone que cada proceso tiene su propio espacio de direcciones (espacio de direcciones virtual del usuario). El kernel tiene su propio espacio de direcciones conocido como espacio de direcciones virtual del sistema. El manejador de dispositivo copia datos entre el espacio de direcciones del kernel y el espacio de direcciones del programa usuario. La transferencia de datos entre la memoria accesible al kernel y el dispositivo es si es dependiente de la maquina. Algunas maquinas requieren que el CPU ejecute instrucciones I/O especiales para mover datos entre un dispositivo y memoria direccionable (llamada acceso directo a memoria DMA). Basicamente lo que se logra con dispositivo manejador es tener un programa que verifica si el hardware está listo para transferir datos. Luego una rutina transfiere una cadena de caracteres de determinada cantidad de bytes desde el espacio de memoria de usuario al dispositivo. Usando interrupciones, el hardware esta permitido de interrumpir

cuando este listo para transferir datos y no haya que esperar.
Los pasos para poner operativo un manejador de dispositivo luego de completar su escritura se pueden consultar en la referencia [26]. Luego de culminada está etapa se procede a realizar las modificaciones correspondientes a la simulación y control del proceso dentro del sistema de software. La parte relacionada a los módulos simulados de los procesos no cambia en terminos orientados a objetos.
Lo necesario a modificar aqui esta en la parte del subsistema Simulador donde los datos que se adquieren para la simulación ya no serán de un algoritmo, sino directamente de la lectura del hardware de la PC, permaneciendo el transito de datos entre subsistemas exactamente igual, variando unicamente la forma de adquisición de datos. Otra detalle a tomar en cuenta es que inicialmente se buscará que el subsistema Simulador se ejecute solamente en la PC donde se implemente la conexión al sistema de hardware del proceso real. Ello para evitarnos en un primer momento las dificultades que se podrían presentar al capturar datos remotamente tomando en cuenta que el tráfico de la red es variable y los procesos reales estan limitados por un periodo de muestreo limitado.

- **Animación gráfica de los procesos**
  El proceso de animación de los procesos en una primera etapa tal vez no se logre en tiempo real de ejecución sino que será un módulo adicional del subsistema Visualizador que logrará via una imagen la sensación de movimiento a partir de la lectura de un archivo de datos con formato predeterminado donde se almacenarán los datos que resultarán de la simulación real del proceso.
  Una aplicación disponible en internet sin costo alguno y que es factible de usar para una aplicación de este tipo es XAnimate, un trabajo divulgado por el Departamento de Ingeniería Eléctrica de la Universidad de Ohio. El incluir el módulo de animación involucra cambios minimos en el software, especificamente en la interface de usuario principal en lo referente al anfitrión donde se ejecutará el módulo. Además en el proceso de simulación se hace necesario crear un archivo con formato prestablecido, que será el que utilizará el módulo de Animación. Un ejemplo de la aplicación se muestra en la figura 7.1

- **Compilación dinámica de módulos**
  Ampliación de los procesos disponibles, mediante una utilidad que nos permita la compilación dinámica de módulos, facilitando la tarea de agregar un nuevo proceso al sistema de software. Luego de tener una primera versión final del sistema de software se desearia que el agregar nuevos procesos capaces de ser simulados sea una labor es más simple posible en lo referente a cambios a realizar. Está idea

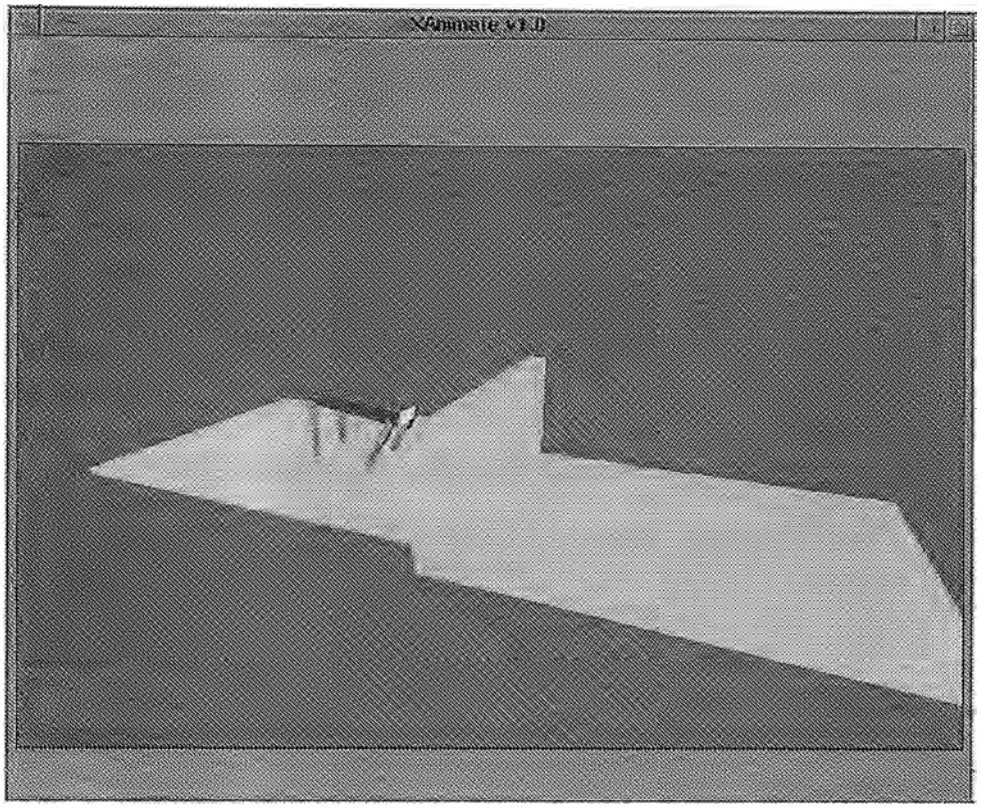

Figura 7.1: Aplicación XAnimate

estará basada en dos criterios, el primero que está repartido a tener un orientado a objetos de tal manera que cualquier proceso a añadir tenga un código escrito de forma natural y no viole las reglas de programación de los demás procesos. El segundo esta relacionado con el hecho de añadir un nuevo proceso sin tener que recompilar todo el software nuevamente, manteniendo asimismo compatibilidad del nuevo proceso con los demás en tiempo de ejecución. En internet se encuentran disponibles varias librerias que ayuden a lograr lo que se propone, ademas de contarse con la información disponible en la distribución de instalación del Linux de como lograr una compilación dinámica. El método consiste en que a partir del código objeto del nuevo proceso este se una al sistema de software en forma dinámica sin necesidad de volver a compilar el sistema de software nuevamente.

BIBLIOGRAFÍA

[1] Douglas M. Considine *Process/Industrial Instruments and Controls Handbook* McGraw-Hill, INC.

[2] *Informix: Manual de Administración* Informix Press

[3] Shreve D. H. y Roberts S. K. *The future of distributed control*, I&CS, vol 59. Nro. 10, setiembre 1986.

[4] Andrew S. Tanenbaum *Redes de Ordenadores* Prentice Hall

[5] Documentos ComputerWorld *Fundamentos Cliente/Servidor* IBM

[6] James R. Groff, Paul N. Weinberg *Aplique SQL* McGraw-Hill

[7] Englewood Cliffs *The Design of the Unix Operating System* Prentice Hall.

[8] Brian W. Kernighan, Dennis M. Ritchie *El Lenguaje de Programación C* Prentice Hall.

[9] *The Linux Bible, The GNU Testament-2nd Edition* Yggdrasil Computing Incorporated, Version 2.1.1 July 1994

[10] *X11 Window System User's Guide* O'Reilly & Associates, Inc.

[11] Richard N. Taylor, Nenad Medvidovic *A Component-and Message-Based Architectural Style for GUI Software* IEEE Transactions on Software Engineering, Vol 22, Nro. 6, June 1996.

[12] G. fischer, A. Girgensohn. *Supporting Software Designers with Integrated Domain-Oriented Design Environments* IEEE Trans. Software Engineering, vol 8, Nro. 6, June 1992.

[13] K. Brockschmidt, *Inside OLE 2* Microsoft Press, 1994

[14] J. Coutaz *Architectual Design for User Interfaces* Proc. Software Engineering, Oct 91.

[15] *Motif* O'Reilly & Associates, Inc.

[16] Katsuhiko Ogata *Ingeniería de Control Moderna* Prentice Hall 1993

[17] Basili V. *Models and Metrics for Software Management and Engineering* IEEE Computer Society Press, 1980, pp 4-9.

[18] Paul L. Meyer *Probabilidad y Aplicaciones Estadisticas* McGraw-Hill

[19] Putman L. *A General Empirical Solution to the Macro Software Sizing and Estimating Problem* IEEE Trans. Software Engineering, vol 4, Nro. 4, 1978, pp 345-361.

[20] Hall. A. *Seven Myths of Formal Methods* IEEE Software, September 1990, pp 11-20.

[21] Roger S. Pressman *Ingenieria del Software* McGraw-Hill

[22] Frank Brokken & Karel Kuba *C++ Annotations (ver 3.3.3)* URL:http://www.iccerug.nl/docs/cplusplus/cplusplus.html

[23] Fco. Javier Ceballos *Curso de Programación C++* Addison Wesley

[24] Hikmet Senay, Eve Ignatius *A Knowledge-Based System for Visualitation Design* IEEE Computer Graphics and Aplications

[25] Dan Heller *XView Programming Manual* O'Reilly & Associates, Inc. Vol. Seven

[26] K. Johnson *Linux Kernel Hacker's Guide*